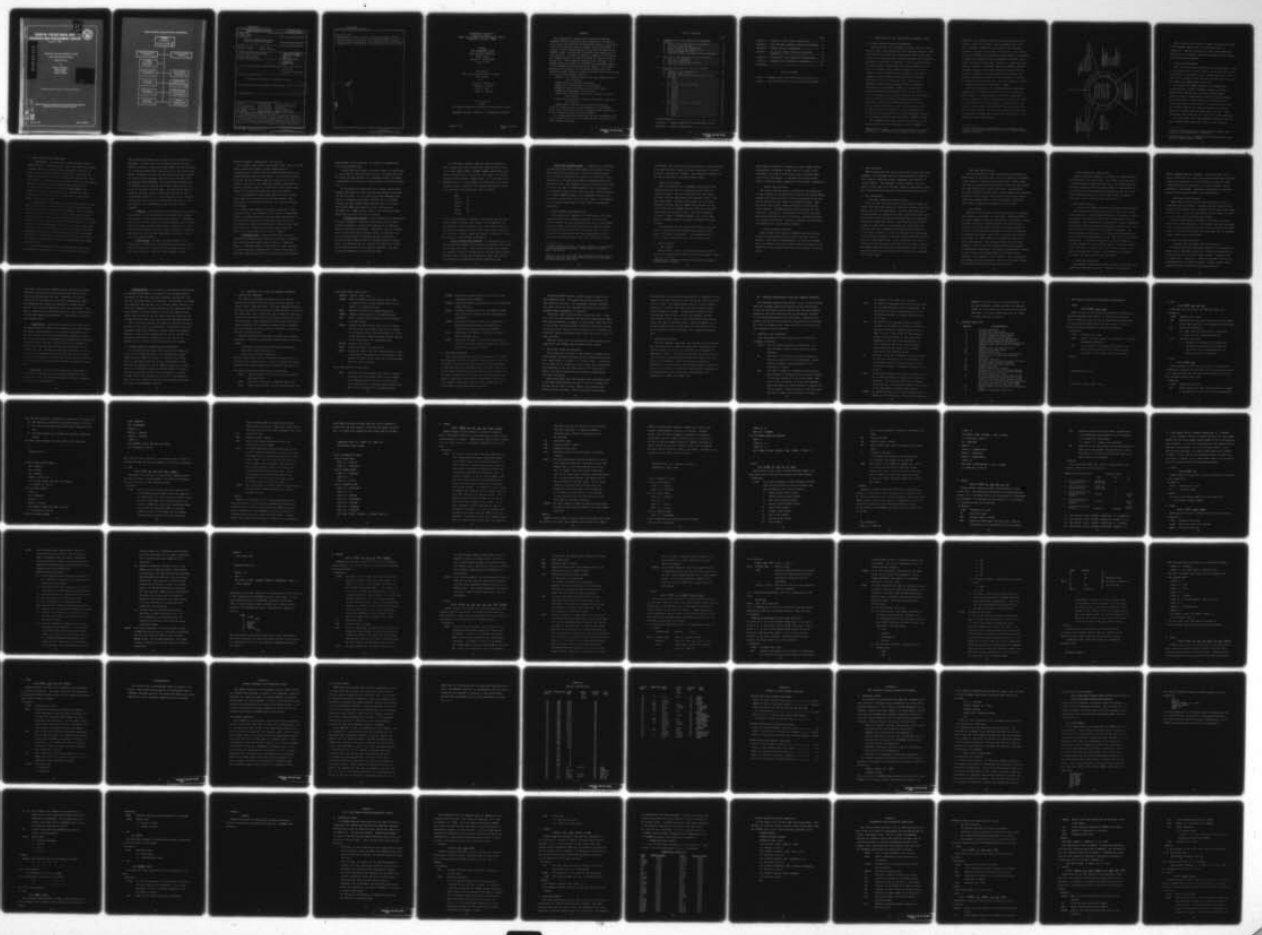
AD-A047 852 DAVID W TAYLOR NAVAL SHIP RESEARCH AND DEVELOPMENT CE--ETC F/G 9/2
COMRADE DATA MANAGEMENT SYSTEM HOST LANGUAGE INTERFACE USERS MA--ETC(U)
JAN 76 W C GORHAM, S E WILLNER, M S SHAW

UNCLASSIFIED DTNSRDC-76-0006 NL

1 OF 2
ADA047852



DAVID W. TAYLOR NAVAL SHIP RESEARCH AND DEVELOPMENT CENTER

Bethesda, Md. 20084

COMRADE DATA MANAGEMENT SYSTEM
HOST LANGUAGE INTERFACE
USERS MANUAL

by

William C. Gorham, Jr.
Stanley E. Willner
Mark S. Shaw
Michael A. Wallace

COMPUTATION, MATHEMATICS AND LOGISTICS DEPARTMENT
RESEARCH AND DEVELOPMENT REPORT

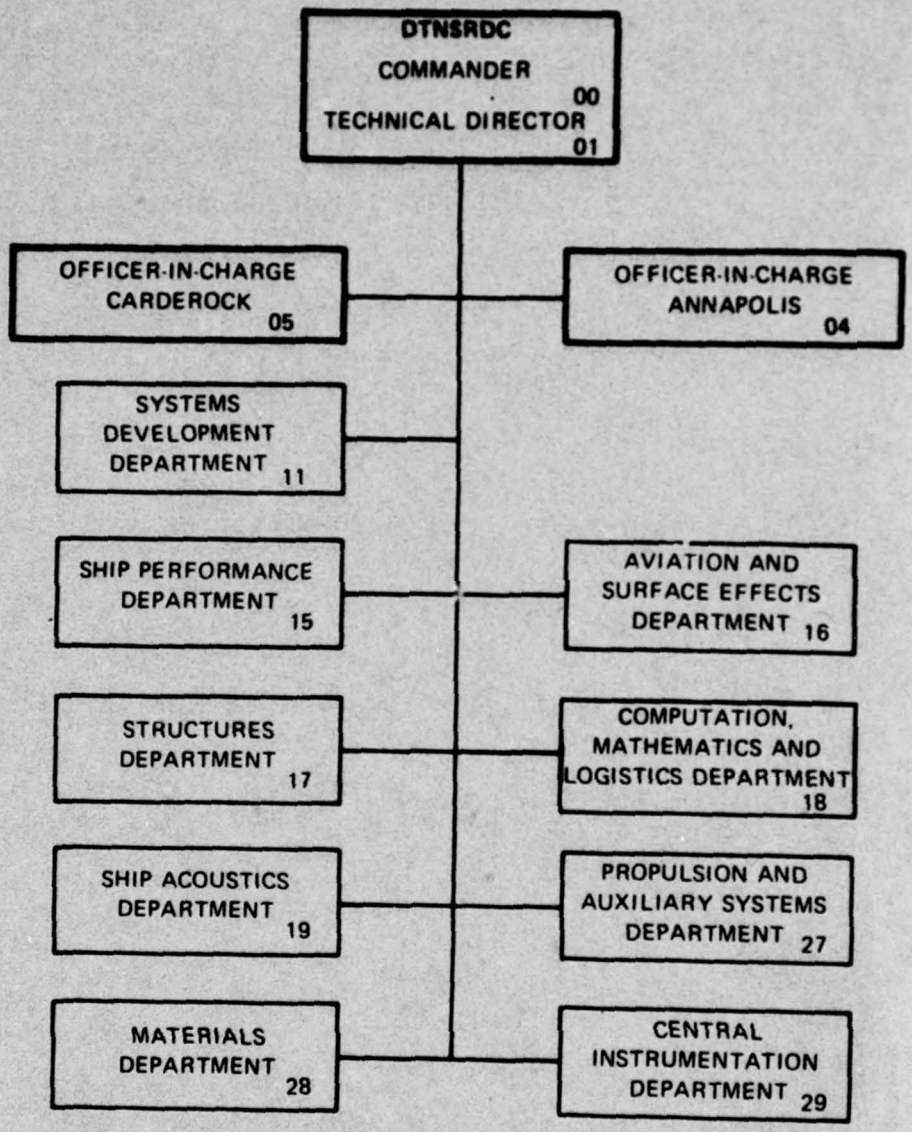
January 1976

Report 76-0006

Report 76-0006

COMRADE DATA MANAGEMENT SYSTEM HOST LANGUAGE INTERFACE
USERS MANUAL

MAJOR DTNSRDC ORGANIZATIONAL COMPONENTS



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: DTNSRDC-76-0006
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle): COMRADE DATA MANAGEMENT SYSTEM HOST LANGUAGE INTERFACE USERS MANUAL
5. TYPE OF REPORT & PERIOD COVERED
6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s): William C. Gorham, Jr. Mark S. Shaw
Stanley E. Willner Michael A. Wallace
8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS: David W. Taylor Naval Ship Research and Development Center, Bethesda, Maryland 20084
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 63509N Project S0331H - Task 14525 Work Unit 1850-030

11. CONTROLLING OFFICE NAME AND ADDRESS
12. REPORT DATE: January 1976
13. NUMBER OF PAGES: 108

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Research and development rpt.
15. SECURITY CLASS. (of this report): UNCLASSIFIED
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

COMRADE	Data definition	Data base administration
Data base management	Data maintenance	Conversational interface
Self-contained system	Report generation	Data dumping
Host language interface	Query processing	Data base statistics
Data loading	Hashing technique	Computer-aided design

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The COMRADE Data Management System (CDMS) is a generalized data base management system that has been developed as an independently useful part of the Computer-Aided Design Environment (COMRADE) System. CDMS consists of two distinct components: a set of host language subroutines which enable CDMS functions to be embedded within FORTRAN Extended and COBOL application programs; and a set of conversational programs which enable a user seated at a

(Continued on reverse side)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-LF-014-6601

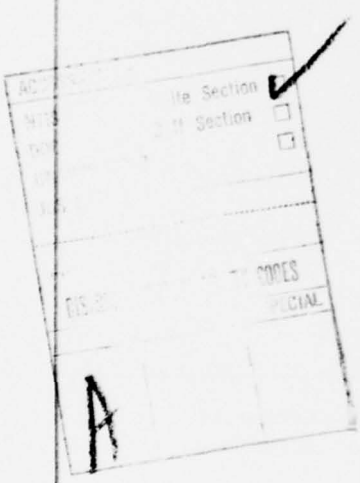
UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

387 682

(Block 20 Continued)

keyboard/printer or display terminal to define, load, update, and query a CDMS data base, to issue a variety of file management commands, and to access a number of auxiliary capabilities. This report provides an overview of CDMS, and describes the procedural steps involved in the use of the CDMS host language interface. CDMS operates on the DTNSRDC Control Data 6700 computer.



DEPARTMENT OF THE NAVY
DAVID W. TAYLOR
NAVAL SHIP RESEARCH AND DEVELOPMENT CENTER
Bethesda, Maryland 20084

COMRADE
DATA MANAGEMENT SYSTEM
HOST LANGUAGE INTERFACE
USERS MANUAL

CASDAC No. 520523/APAC

January 1976

Prepared by
The Naval Ship Engineering Center
Code 6105B
by
William C. Gorham, Jr.
Stanley E. Willner
Mark S. Shaw
Michael A. Wallace

Developed for
CASDAC
The Computer-Aided Ship Design and Construction Project

January 1976

Report 76-0006
UM-05

FOREWORD

The comprehensive Computer-Aided Design Environment (COMRADE) software system has been developed at the David W. Taylor Naval Ship Research and Development Center, sponsored by the Naval Sea Systems Command (NAVSEA, formerly NAVSHIPS) in support of their Computer-Aided Ship Design and Construction (CASDAC) effort to facilitate the design and construction of Naval vehicles. Although conceived in support of ship design, the COMRADE software has been developed for use as a general design tool. Three separate components are included: the COMRADE Executive System; the COMRADE Data Management System; and the COMRADE Design Administration System. All of these components are operable on the DTNSRDC Control Data 6700 computing system (SCOPE 3.4 operating system) and are documented in a set of eight DTSNRDC reports:

COMRADE - The Computer-Aided Design Environment Project, An Introduction

COMRADE Executive System Users Manual

COMRADE Data Storage Facility Users Manual

COMRADE Absolute Subroutine Utility Users Manual

COMRADE Data Management System Primer

COMRADE Data Management System Host Language Interface Users Manual

COMRADE Data Management System Conversational Interface Users Manual

COMRADE Design Administration System Users Manual

Inquiries concerning any of the components of the COMRADE system should be directed to the Computer Systems Development Group, Computer Sciences Division, DTNSRDC.

It is understood and agreed that the U.S. Government shall not be liable for any loss or damage incurred from the use of these computer programs.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION TO THE COMRADE DATA MANAGEMENT SYSTEM

A. SYSTEM CLASSIFICATION AND ORIENTATION

Generalized data base management systems might be classified as either self-contained or host language interface types. A self-contained system typically provides generalized capabilities for data definition, data maintenance, data retrieval, and report formatting. In addition, a self-contained system usually provides a generalized retrieval language that can be used to express fairly complex selection criteria. In contrast, a host language system provides a generalized capability only for data definition, and an interface, composed of a set of high-level calls, which may be embedded in application programs written in procedure languages such as FORTRAN or COBOL. It is via these specialized application programs that the end user of a host language data base system performs data maintenance, query processing, and report formatting procedures. Although an application program interface to the data base is always provided by a host language system, such an interface may be included in a self-contained type of system as has been done in the data management component of the Computer-Aided Design Environment (COMRADE) System.

The COMRADE system,[1] developed at the David W. Taylor Naval Ship Research and Development Center (DTNSRDC) in

[1] Rhodes, T., "COMRADE - The Computer-Aided Design Environment Project - An Introduction," DTNSRDC Report 76-0001.

support of the Computer-Aided Ship Design and Construction (CASDAC) Project,[2] provides a self-contained type of data base management system which includes the host language interface described in the pages that follow. This system, called the COMRADE Data Management System (CDMS), is designed to enable non-computer-oriented personnel to define, load, update, and query a data base. Moreover, it supports a host language interface to FORTRAN, COBOL, and COMPASS assembly language programs so that when specialized needs arise--as frequently happens with CASDAC application software, for example--application programs can be developed to interact as necessary with the data base. The host language interface is therefore an important component of CDMS.

CDMS was developed in the FORTRAN Extended programming language and is intended primarily for use in scientific applications. It was designed to facilitate management of numeric information used in applications involving the design and construction of ships, aircraft, buildings, dams, and bridges, among others. Although CDMS does provide for the use of textual values, it does not facilitate the query and maintenance of textual data as would be required for bibliographic and legal applications.

---

[2] "Naval Ship Systems Command Technical Development Plan - Computer-Aided Ship Design and Construction," NAVSHIPS Report S46-33X (Feb 1970).

Figure 1 - DTNSRDC Control Data 6700 Computing Facility

CDMS is operational under the SCOPE 3.4 operating system[3] on the DTNSRDC Control Data 6700 computing facility (Figure 1). CDMS was developed by, and is maintained and enhanced by, the Computer Systems Development Group, Computer Sciences Division, DTNSRDC.

B. TRI-LEVEL ARCHITECTURE

CDMS is hierarchically structured into three levels. Users are permitted access to their data at any of the three levels. The flexibility provided by such a design permits the individual user to make his own trade-offs between ease of use of the system and computational efficiency.

The lowest-level CDMS component, the COMRADE Data Storage Facility (CDSF),[4] supports storage, retrieval, modification, and deletion of named variable-length logical records or data blocks, and the optional building and processing of inverted data lists for retrieval of information based on data values. The subroutines of CDSF are generally used by the system programmer who values economy of computer time and space above all else. CDSF constitutes the foundation for higher-level CDMS components.

The next higher level CDMS component comprises a set of host language subroutines which serve as the interface

---

[3] Control Data Corporation, "SCOPE Reference Manual, 6000 Version 3.4," Publication No. 60307200.

[4] Wallace, M., et al., "COMRADE Data Storage Facility Users Manual," DTNSRDC Report 76-0003.

between an application program's data requirements and CDSF. It is at this level that logical items of information and groups of information may be retrieved and updated by name. Information may also be retrieved by way of queries involving keyed data attributes. Variable-length logical records may be defined as groups of single-valued elements, arrays, and repeating groups of elements. Pointer-type elements may be used to link records to form a variety of logical data structures.

The third and highest level CDMS component is provided to make the system most usable to non-computer-oriented personnel. This component comprises a set of modules suitable for use in batch mode but designed primarily for conversational use.[5] In the conversational mode, a user seated at a keyboard/printer or display terminal may define, load, update, and interrogate his data base. As with the host language interface, the conversational interface allows a user to work with logical items of information. While performing data base retrieval, a user specifies his selection criteria in English-like phrases. All of the CDMS conversational commands are available to any valid user of DTNSRDC's INTERCOM time-sharing system.[6]

The performance of data base definition and processing at each of the three levels of CDMS is depicted in Figure 2.

---

[5]Gorham, W., et al., "COMRADE Data Management System Conversational Interface Users Manual," DTNSRDC Report 76-0007.

[6]Control Data Coporation, "INTERCOM Reference Manual, 6000 Version 4," Publication No. 60307100.

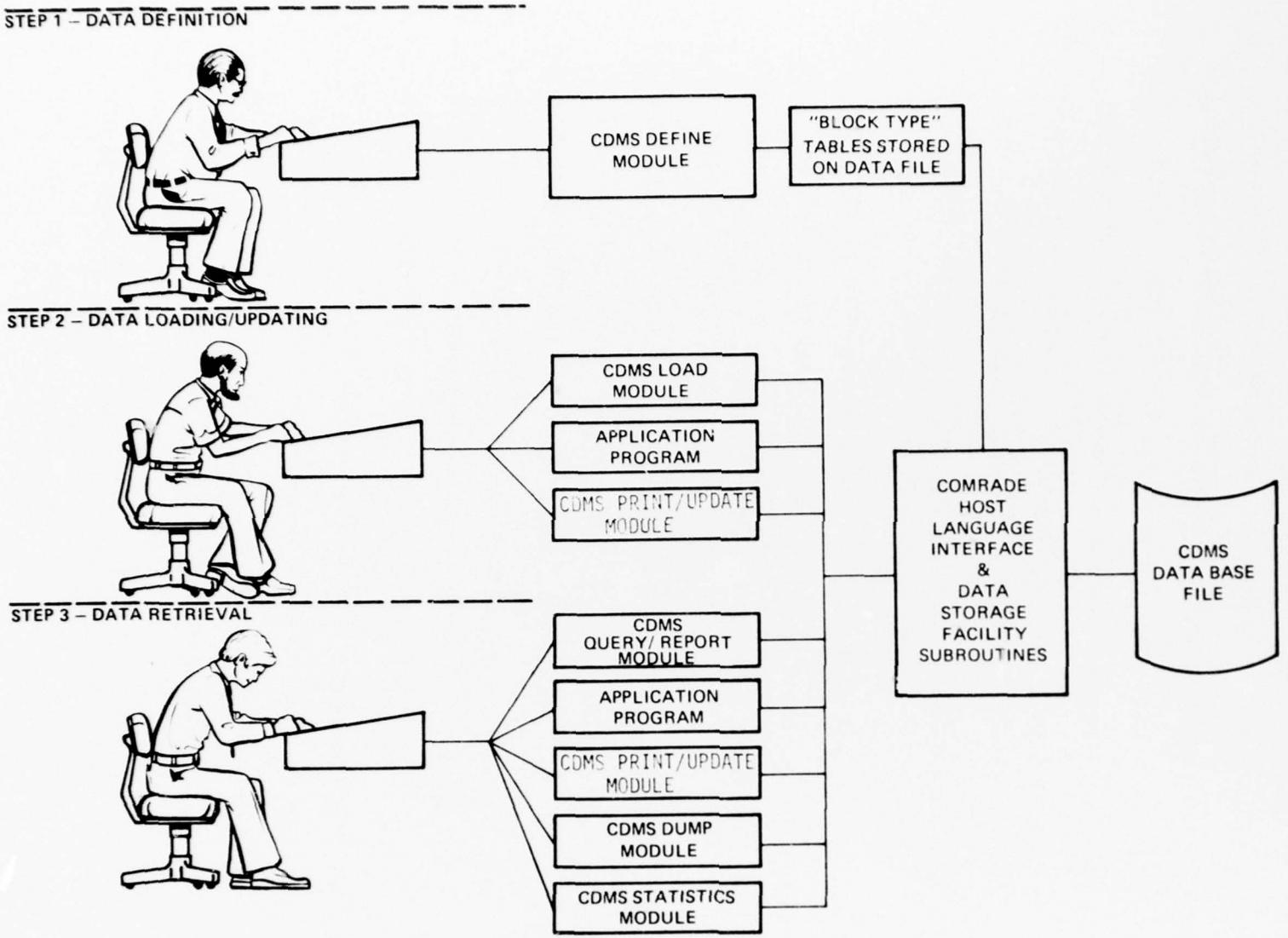


Figure 2 - Data Base Definition and Processing Using CDMS

C. BASIC CONCEPTS AND TERMINOLOGY

. DATA BLOCK. The basic unit of data storage within a CDMS data base is the data block. Each data block is assigned a unique name of as many as eight characters, the first of which must be a letter, which is used in all retrieval and update transactions involving information in that block. Thus, for example, in a data base of information about U.S. Presidents, data pertaining to John F. Kennedy might conceivably be collected into a data block given the name KENNEDY; and similarily, within a ship design file, information about a particular deck would likely be grouped into a data block given a name such as DK03.

. BLOCK TYPE. A CDMS data base file comprises one or more sets of data blocks. Each set has a unique record format or definition called a "block type" description. This description gives the logical structure of all data blocks of that particular block type. The description defines subblocks, repeating groups, and data elements and defines the data type (i.e., single- or multi-valued, alphanumeric, pointer, real, and integer) and inversion status of each element. A U.S. Presidents data base, for example, might have 38 data blocks of the block type PRES (personal information about the Presidents) and 48 data blocks of the block type ELECTION (information pertaining to presidential elections).

To further illustrate the block type concept, assume that the personnel department of a large installation has a data

base containing personal data on each of the installation's employees. Personal data for an employee named Tom Jones might be stored in a data block names JONES, and personal data for an employee named John Smith stored in a data block named SMITH. Although this data base may include many data blocks, all blocks storing personal data will be of the same block type perhaps named EMPLOYEE. The block-type description for the EMPLOYEE blocks indicates the kinds of data values contained in that set of data blocks. The description may, for instance, indicate that the third data value in the data blocks JONES, SMITH, etc., is the integer value for the data element AGE, and that the fifth data value in each of these data blocks is the alphanumeric value for the element BIRTHDAY.

. SUBBLOCK. For convenience in organizing and retrieving data elements, a data block can be subdivided into uniquely named groups of data elements known as subblocks. Subblocks may be used to group logically related elements and to provide access to subsets of data values within a data block. Each data block may contain as many as eight subblocks. A subblock name may be as many as eight alphanumeric characters long, the first always a letter.

. DATA ELEMENT. In CDMS, the data element is the smallest unit of data that can be defined and referred to by name (the name to be as many as eight characters long, with the first a letter). There are three classes of data

elements--numeric, alphanumeric, and pointer:

(a) A numeric data element may contain either real or integer values, and may be either a single-valued quantity (represented by a single 60-bit CDC 6000 computer word) or a variable-length array of such values. If an array of values is given, the data element name will refer to all of the values in the array. The CDMS host language subroutines do, however, provide a means of referring to a particular array element if its position within the array is provided.

(b) An alphanumeric data element may be either a single value (made up of as many as ten CDC 6000 display code characters), an array of such values, or a variable-length text string.

(c) Pointer data elements are used to construct a data structure (tree, ring, network) that links all related data blocks. A pointer element in one data block may "point" to a different data block on the same data base file, or it may "point" to a data block stored on a different data base file entirely. A pointer data element may be defined either as a single value or an array of values.

. REPEATING GROUP. Within a data block, one to 50 consecutively-defined data elements of the same or different type may be grouped together and defined as a repeating group. A repeating group is identified by a name (as many as eight characters, beginning with a letter) which may be used when referring to the entire grouping of data elements.

Each element in the group may, of course, be referenced by its own individual name.

A data block may contain a variable number of repetitions or occurrences of repeating group values. Although there may be at most 50 data elements in a repeating group, the number of occurrences of values for these elements is virtually limitless.

To illustrate the repeating group concept, assume that employee Tom Jones has three children and employee John Smith has but one child. If in a block type named EMPLOYEE there exists a repeating group named CHILDREN, and if CHILDREN includes the data elements NAME and AGE, then in the data block named JONES there would be three occurrences of CHILDREN (i.e., NAME and AGE) while in the data block named SMITH there would be only one occurrence.

. INVERTED DATA ELEMENT. Quick resolution of conditional retrieval requests or "queries" issued from either conversational terminals or application programs is an important aspect of data base processing. CDMS uses a set of "inverted lists" to minimize query response times. An inverted list is a directory which (1) contains the data values for a particular data element which was assigned inversion status when it was defined, and (2) contains the names of all data blocks having values for the inverted elements. The data-value/block-name pairs are sorted by value.

To illustrate, assume a CDMS data base of personnel data. Assume also that the data base contains data blocks of the names JONES, SMITH, JOHNSON, BROWN, and WILSON, which contain personal data on Tom Jones, John Smith, Jim Johnson, Ed Brown and Bob Wilson, respectively. If in the block type EMPLOYEE there is a data element AGE which has been assigned inversion status, values for the element AGE will be stored in an inverted list which might look as follows:

AGE	
	25
Smith	
	28
Johnson	
	28
Brown	
	36
Jones	
	38
Wilson	

If the user requested a printout of the values for the data element WEIGHT (another data element within EMPLOYEE) for all persons at least 30 years of age, the inverted list for AGE would be examined, and only those data blocks satisfying the condition - namely, JONES and WILSON - selected for further processing to obtain WEIGHT information.

. LOCALLY DEFINED DATA ELEMENTS. An individual data block may include data elements not logically related to other blocks of the same block type. These locally defined data elements are managed in exactly the same way as the other data elements defined within a block type description except that they are not considered for membership within repeating groups.

. CROSS FILE REFERENCE TABLE. In addition to containing user-defined data blocks, block types, and inverted lists, a CDMS data base file contains a cross-file reference table which lists the permanent file names of all data base files "pointed to" by pointer data elements contained within this particular file. The permanent files listed in the table may be one of two types: a SCOPE permanent file, indicated by a four-character ID and a file name of as many as 40 characters, or a CPFMS (COMRADE Permanent File Management System) file, indicated by level-three and level-four names. (See Appendix A for a detailed description of the CPFMS file management capability.)

D. SYSTEM FEATURES AND CAPABILITIES

To facilitate the comparison of CDMS features with other generalized data base management systems, the following feature analysis has been patterned after that of the CODASYL Technical Report of May 1971[7] and after the National Bureau of Standards Technical Note 887 of November 1975 (a comparison of six generalized data base management systems).[8] Features are described under eight major headings: computer

---

[7] CODASYL Systems Committee, "Feature Analysis of Generalized Data Base Management Systems," CODASYL Systems Committee Report (May 1971).

[8] Fong, E., et al., "Six Data Base Management Systems: Feature Analysis and User Experience," National Bureau of Standards Technical Note 887 (Nov 1975).

environment, data structures, data definition, data maintenance, query specification, output and report generation, security features, and external linkages. Each heading is further subdivided as necessary to provide greater detail.

1. Computer Environment

CDMS is operational on the DTNSRDC Control Data 6700 computing facility which is supported by the SCOPE 3.4 operating system. The core requirement for the conversational interface approximates 25K (decimal) words. The core requirement, excluding buffer space, for the host language interface and the COMRADE Data Storage Facility (when used with the COMRADE Absolute Subroutine Utility)[9] approximates 8K (decimal) words. The source language for most of CDMS is FORTRAN Extended; only a very small number of COMPASS assembly language statements have been included.

The communication link between the terminal user and CDMS is supported by the INTERCOM time-sharing system. CDMS is not coded re-entrantly. Its components may be used in either the batch or the conversational mode.

2. Data Structures

. Data Types

CDMS provides a set of valid data types which can be defined for single-valued elements and array elements. Those

---

[9]Wallace, M., "Absolute Subroutine Utility Users Manual," DTNSRDC Report 76-0004.

valid types are numeric, including real and integer values; alphanumeric, including a single value of as many as ten characters, an array of such values, and a variable-length text string; and pointer. A character may be any letter, digit, or special symbol supported by the hardware (Appendix B).

. Logical Data Structure

The logical data structure is defined as the composition of data elements in the data blocks, and the interrelationships between data blocks and their files as determined by the user without regard for main memory or secondary storage. CDMS data blocks may contain single-valued and variable-length array elements. Variable-length repeating groups are also permitted. Data may be interrelated across files and among data blocks through the use of "pointers", thereby providing full network structuring. Moreover, inverted list structures are supported for single-valued elements.

. Physical Storage Structure

Data records within CDMS are randomly organized and are located by transforming the data block name into the disk address. This type of transformation, or hashing technique, makes it possible to access a block during its residence on a CDMS file by its symbolic name rather than through a numerical index.

3. Data Definition

CDMS provides the user with a block type or data definition language. This language serves as the mechanism for defining data elements, repeating groups, subblocks, and keyed or inverted elements. The language is card oriented - one card image for each item to be defined. Within a card image, fields may be specified free-form with commas separating the fields.

4. Data Maintenance

. Data Base Loading and Input Analysis

After the user has described his block types to CDMS, the system uses these descriptions to construct internal block-type tables, one for each description. At data base load time, a block-type table is identified by name, after which one or more data blocks of this particular block type may be created and, optionally, loaded with values. After a data block has been created, a value that is to be loaded into that block must first be read from a sequentially organized card image file. If the value is numeric, it will be converted into the proper internal format pursuant to the data element definition stored within the block-type table. After conversion, the value is located into the data block and referred to in future retrieval and update requests simply by its data element name. A data definition links the card image file of values to a particular set of block types.

. Data Base Restructuring

The restructuring of a data base generally involves the modification of the block type, such as by adding or deleting a data element or repeating group definition. Within CDMS, data base restructuring is permitted so long as no data blocks have been created for a given block type. Restructuring following creation of data blocks involves dumping the existing contents of these blocks, redefining the block type, and reloading the dumped values. A utility program is available under CDMS for dumping the contents of a data base.

. Data Updates

Updating is the process of adding, modifying, or deleting the contents of some part of a data base. Updating differs from data loading in that the load process is typically a batch procedure involving a sizable quantity of values and the creation of new data blocks, whereas the update process is typically a conversational procedure involving transactions on existing data blocks and the use of fairly small sets of data values. In CDMS, there is a language for specifying update transactions. The language differs from the query language in that it does not involve English-like phrases and conditional operations. The update language does, however, provide for both the updating and the printing of data elements. The language is card oriented - one card image for each update or print operation. Within a card image, fields may be specified free-form separated by slashes.

. Other Maintenance Capabilities

Lockout during the updating or loading of a CDMS data base is maintained at the file level. As a result, only one user at a time may modify a CDMS data base file. To minimize possible conflicts resulting from concurrent update or load operations, the scheduling of data base modifications may have to be controlled administratively. During a modification procedure, users may retrieve and print data base values.

5. Query Specification

. Overall User-System Interaction and Search Specification

Query processing involves the selective retrieval of data from a data base. Within CDMS, information is entered initially to identify the particular portion of the data base to be interrogated. That information is followed by a set of statements that specify selection criteria and certain actions to be taken such as print in columnar or row format, for example. The query language is based upon use of English-like phrases which may contain Boolean combinations (i.e., AND or OR) of relational expressions. A relational expression in most instances is a triple composed of a data element name, a relational operator (i.e., GT, GE, LT, LE, NE or EQ), and one data value; in the case of the operator "between", the expression is composed of a name, the operator BET, and two values.

. Multi-File Searching

The CDMS Query capability enables as many as five data base files to be searched initially. Cross-file and intrafile

pointer elements may be traversed. With the current "hit" file (a "hit" file is a list of data blocks which have satisfied a specific condition) as a starting point, as many as eight different pointers (specified in a single statement) may be processed by the system. Typically, processing of cross-file pointers will continue without user intervention.

. Additional Query Features

CDMS allows hit files to be named and saved for use in subsequent query statements. Searches based on values of non-inverted data elements may be conducted on the data blocks named in a hit file. A hit file name table, and the number of "hits" on a hit file, will be printed upon request.

A pseudo batch mode is available to the user during conversational query processing. In this mode, the user may submit a set of query statements which will then be checked for syntax errors; if there are no errors, the statements will be executed without further user intervention.

6. Output and Report Generation

. Language Type and Media Selection Flexibility

An output and report generation capability is provided as a subset of the CDMS Query capability. English-like phrasing is used to specify one of three report formats: columnar, row, or simple list. Typically, a report is printed on the user's keyboard/printer or display terminal. Lengthy reports can be routed to off-line medium and high speed printers.

. Arithmetic and Sorting Capabilities

Computations on retrieved numeric values are supported within CDMS by a set of built-in functions which include sum, difference, quotient, product and columnar totals, and averages. Sorting, also supported by a built-in function, may be specified in columnar-report requests and may be performed on as many as eight fields in accordance with the order indicated (i.e., LOW A, HIGH B, LOW C, etc.).

7. Security Features

. File Protection and Access Control

A CDMS data base file must be given a name (of as many as 40 characters), and may be password protected in accordance with SCOPE 3.4 operating system conventions. The file is protected from unauthorized access by way of a set of privacy controls specified at the time the file is cataloged into the SCOPE system. Since access priveleges may vary from user to user, a variety of controls is provided including read only, read and modify, increase mass storage allocation, and delete a file from the SCOPE file system.

An additional level of CDMS data base file protection may be imposed by way of the COMRADE Permanent File Management System (CPFMS). A detailed description of the CPFMS capability is provided in Appendix A.

. File Backup, Recovery, and Restart

At DTNSRDC, safeguards exist to prevent an on-line CDMS file

from being accidentally destroyed during normal system operation and normal loading of the operating system. For backup purposes and to guard against data being lost as a result of some abnormal system event, on-line files are generally dumped to magnetic tape, on a semi-weekly basis. A variety of utility programs exists at DTNSRDC for use in more frequent dumping of data base files. Among these is the CDMS conversational interface DUMP module.

8. External Linkages

As mentioned previously, CDMS supports a host language interface to COBOL[10], FORTRAN Extended[11], and COMPASS[12] applications programs. These programs request services of the host language subroutines through control statements such as the CALL statement of FORTRAN Extended, for example.

E. DATA BASE ADMINISTRATION

In an environment such as computer-aided ship design, in which the data base includes data to be shared by many users and programs, it becomes necessary for certain functions to be centrally coordinated and controlled. In such an environment the data base must inevitably reflect a compromise between

---

[10]Control Data Corporation, "COBOL Version 4 Reference Manual," Publication No. 60384100.

[11]Control Data Corporation, "FORTRAN Extended Version 4 Reference Manual," Publication No. 6035601.

[12]Control Data Corporation, "COMPASS Version 3 Reference Manual," Publication No. 60360900.

the needs of the various competing users and their programs. Conflicts resulting from such diverse needs must be satisfactorily mediated and resolved. Generally this role of arbitrator is performed by a data base administrator (DBA) who possesses data processing experience, communications skills, and a working familiarity with user operations. The DBA is responsible for functions spanning three broad categories of data base design and management: organization, monitoring, and reorganization.

. ORGANIZATION. During the organization of a data base the DBA assesses the needs of the competing users and programs. Following a determination of all the data requirements, the DBA will (a) use the block-type definition language to define the logical structure of the data base including those subblocks, repeating groups, and data elements that serve to model the problem with which the data base will be concerned; (b) coordinate data base loading and updating activities; (c) assign data base file names and access controls, taking into account the access privileges of all data base users; and (d) coordinate the activities of any specialists working to organize the data base.

. MONITORING. During the life cycle of a data base, it is the DBA's responsibility to develop strategies for monitoring the use of the data base, breach of privacy, frequency of backup, and reorganization.

. REORGANIZATION. As a result of information obtained from the monitoring procedure, or because of the changing data requirements of the users and their programs, the DBA may find that the data base must be reorganized. The DBA may: (a) change the block-type descriptions, (b) re-assign access controls, (c) alter the frequency of backup operations, (d) change the data base contents to reflect changes in the block types and the logical data structure, and (e) remove from the data base any allocated storage space not currently being used productively.

These activities indicate the kinds of functions for which the DBA and his staff are responsible. Since many of the user languages supported by CDMS are oriented toward non-computer personnel, they are most appropriate for use by the DBA and persons working under his direction in that they allow operations to be readily formulated and concisely stated.

F. POSSIBLE COURSE OF FURTHER SYSTEM DEVELOPMENT

As the computer-aided ship design systems of CASDAC progress from the planning stage to the developmental stage, augmentation of CDMS will likely be required. This effort would be undertaken along two fronts: enhancement of CDMS features and capabilities of the DTNSRDC computing facility, which is to eventually become an operating mode of the Navy laboratory computing network; and development of a subset of CDMS features and capabilities on a set of minicomputers which are to be located at the Naval Ship Engineering Center and linked to the DTNSRDC computer.

II. OVERVIEW OF THE CDMS HOST LANGUAGE INTERFACE

A. ROLE OF THE INTERFACE

The CDMS host language interface serves to link an application program--a ship or aircraft design program, for example--with a CDMS data base. The host language interface enables the program to create sets of data blocks, to retrieve and update data block information, and to query the data base, selectively retrieving data that satisfies certain user specified conditions. When combined with the control and input/output constructs of a procedure language such as FORTRAN, the host language interface provides a powerful, responsive, data base processing tool.

Nineteen subroutines constitute the CDMS host language interface. These may be subdivided functionionally into three groups:

- Data base initialization/finalization subroutines
- Data block update subroutines
- Data block retrieval subroutines

Specific subroutines and the functions they perform are indicated in the following list. Details regarding subroutine operation and calling formats are discussed in Section III.

Data Base Initialization/Finalization Subroutines

SBUFF	Defines a buffer for use in retrieval and update operations.
ILFN	Opens and initializes a CDMS data base file
FLFN	Closes and finalizes the current data base file

Data Block Update Subroutines

DEFINB Creates a data block

CHANGE Adds or changes the data value(s) for either a single-valued or array element that is not a member of a repeating group

ADDO Adds an occurrence to a repeating group

CHANGO Changes an occurrence of a repeating group

ADDU Adds a locally defined element and its value(s) to a data block

DELETE Deletes the data value(s) for either a single-valued or an array element that is not a member of a repeating group; can also delete one or all occurrences of a repeating group element, and one or all occurrences of a repeating group

DELETB Deletes a data block

MODBN Changes the name of a data block

PFNIN Stores the name of the file referenced by a cross-file pointer and completes the definition of that pointer by inserting an index that indicates where the file name is stored.

Data Block Retrieval Subroutines

FETCH Retrieves the data value(s) for either a single-valued or an array element that is not a member of a repeating group; can also retrieve one or all occurrences of a repeating group element, and one or all occurrences of a repeating group.

FETCHN Retrieves the data value(s) for a set of non-repeating group elements.

FETCHR Retrieves the data value(s) for a set of repeating group elements.

QUERY Searches inverted lists for the names of those data blocks that satisfy certain user-specified conditions.

FETCHC Retrieves the data value(s) for a specific repeating group occurrence, according to an identifier provided by subroutine QUERY.

NOCC Determines the number of occurrences within a given repeating group

INPFN Retrieves the name of the file referenced by a cross-file pointer, using an index that indicates where the file name is stored

B. DATA BASE PROCESSING

The initialization/finalization subroutines of the CDMS host language interface are used to define a buffer that will be used for retrieving and updating data base information, to open and initialize the data base for storage and access, and to close and finalize the data base when its current use is ended or another file is to be used for processing. Any number of data base files may be established for use by the host language subroutines subject to the conventions and limitations of the SCOPE operating system.

Subroutine SBUFF defines a buffer which in reality is a user-supplied array. The buffer may contain as many as 20 data blocks at one time. The programmer need not reference this area, since the host language subroutines will perform whatever buffer management is required.

Subroutine ILFN is used to open a data base file. Files will exist on those permanent file devices recognized by SCOPE. When data processing is complete or when another data base file is to be used, subroutine FLFN is called to close and finalize the current file. When the file is closed, it is logically disconnected; further processing cannot take place until a call to ILFN is issued.

CDMS data base files are processed as random files and are supported by the COMRADE Data Storage Facility (CDSF).

C. DATA BLOCK UPDATE AND RETRIEVAL

The storage and retrieval of data blocks is managed by the host language interface software and its underlying primitives. When a data block is created, it is given a name. A directory of the names of all blocks on a file is kept, and includes the disk addresses on the file where each of the blocks may be found. The CDSF is responsible for directory handling and disk space allocation.

After block-types have been defined using the DEFINE module of the CDMS conversational interface [5], the host language subroutines can be used to create, update, and retrieve data blocks and their contents. These subroutines allow named

data elements to be processed symbolically or logically, without regard to internal data block structure or the location of data within a block. The named elements can be of real, integer, character, or pointer type data, and can be either single-valued elements or arrays. Single-valued elements can be inverted and used as keys in retrieval requests; pointer-type elements can be used to form relationships among blocks within the same or different files. Sets of elements can be grouped together under a group name with occurrences of the group repeated as often as needed (i.e., repeating groups of data).

D. AUXILIARY FACILITIES

The host language subroutines are resident on two libraries that are shared by the CDMS user community. Operational procedures relative to use of the libraries are discussed in Appendixes D and F. These shared libraries also contain three sets of utility subroutines which have been found to be quite useful to programmers. These subroutines--discussed in Appendixes E, G and H--allow an executing program to perform a variety of functions, including bit manipulations, alphanumeric data manipulations and conversions, and permanent file management for SCOPE and CPFMS files.

III. DETAILED DESCRIPTION OF THE HOST LANGUAGE INTERFACE

The following detailed descriptions of the 19 user-callable CDMS host language subroutines observe certain conventions. In the calling formats for the subroutines, the user-supplied parameters have been underlined and the parameters returned to the calling program have been overlined. Parameters frequently used by the subroutines are indicated and are followed by a list of standard error codes returned to the user by the host language interface software.

A. FREQUENTLY USED PARAMETERS

Eight parameters are used frequently by the CDMS host language subroutines:

ICODE — An error code returned by a subroutine. In a few instances, ICODE is also used as an input parameter.

BN — The name of a data block on which an update or retrieval operation is to be performed. BN may contain as many as eight characters, left-justified, H format.

SBN — The name or number of a subblock containing the data element or repeating group on which an update or retrieval operation is to be performed. SBN (set to zero if unused) is an optional parameter which can reduce the time necessary to process the user's request. SBN may contain either characters (as many as eight, left-justified, H format), or

an integer in the range 1-8, inclusive.

RGN — The name of a repeating group on which the update or retrieval operation is to be performed. RGN may contain as many as eight characters, left-justified, H format; if unused, RGN should be set at zero.

RGO — The number of the repeating group occurrence involved in the update or retrieval operation. RGO should contain an integer greater than or equal to zero; a zero value for RGO indicates either "all occurrences" or "not applicable" depending on the use of the subroutine. In a few instances, RGO is an output parameter indicating the total number of occurrences contained in the repeating group.

EN — The name of the data element on which the update or retrieval operation is to be performed. EN may contain as many as eight characters, left-justified, H format. If unused, EN should be set to zero.

DATA — As an input parameter, an array of new data values. As an output parameter, an array of retrieved information. The format of DATA differs somewhat from subroutine to subroutine; details are provided in the individual subroutine descriptions.

LENGTH — For update operations, an integer specifying the length of the DATA array. For retrieval operations,

LENGTH is both an input and output parameter. As an input parameter, LENGTH indicates the maximum number of values which may be retrieved and placed into DATA; on output, LENGTH specifies the number of values actually retrieved.

B. STANDARD ERROR CODES

Code No.	Interpretation
0	No error detected
1	No data base initialized
2	Illegal data block specification
3	No value loaded for data element
4	Illegal subblock specification
5	Illegal repeating group specification
6	Illegal data element specification
7	Subscript specification exceeds array size
8	Internal error (may be due to bad data block)
9	Illegal block type specification
10	Data block already exists
11	Illegal buffer size
12	Working area too small for those values which are to be retrieved; no values retrieved
13	Unable to open file
14	No buffer initialized, or buffer too small for processing inverted elements
15	Unable to finalize data base file
16	Illegal data type
17	Illegal logical unit number
18	Syntax error in the array containing the retrieval criteria for subroutine QUERY, or the array is too long
19	An entry containing a block name and the related repeating group occurrence identifiers is too large to be written to the hit file; file unloaded (the user receiving this error may wish to use the COMRADE Data Storage Facility Subroutine CULL)
20	Illegal index into the cross-file reference table
21	Cross-file reference table full

C. DATA BASE INITIALIZATION/FINALIZATION SUBROUTINES

1. SBUFF

SBUFF(ICODE, BUFF, SIZE)

SBUFF defines the data buffer to be used by the CDMS host language interface subroutines. A buffer must be defined before a CDMS data base file can be initialized and processed. If a previously defined buffer has been in use, it will be emptied and made available to the calling program.

Parameters:

ICODE — Standard error code.

BUFF — Name of the buffer space (as declared in a FORTRAN DIMENSION statement[11]).

SIZE — Size of buffer; a minimum of 129 words is required unless inverted lists, which must have a minimum of 1025 words, are to be accessed.

Example:

```
.
.
.
.
.
DIMENSION BUFF (1025)
.
.
.
.
CALL SBUFF (ICODE, BUFF, 1025)
.
.
.
.
```

2. ILFN

ILFN (ICODE, LFN, MR, INV)

ILFN opens and initializes the CDMS data base file.

Parameters:

ICODE Standard error code.

LFN SCOPE logical file name (left-justified, H format) of the data base file to be opened and initialized.

MR Multi-read indicator:

= 0 Open file for read and write processing

≠ 0 Open file for read processing only

INV Inversion indicator:

= 0 By-pass all inverted list transactions

≠ 0 Construct an inverted list transaction file which will be processed when the data base file is finalized.

3. FLFN

FLFN (ICODE, LFN)

FLFN processes the file (if one exists) of inverted list transactions; rewrites the data base file's directory and cross-file reference table (if they have been modified); and closes and finalizes the data base file.

Parameters:

ICODE Standard error code.

LFN SCOPE logical file name (left-justified, H format) of the data base file to be closed and finalized.

D. DATA BLOCK UPDATE SUBROUTINES

1. DEFINB

DEFINB(ICODE, BN, BT)

DEFINB creates a data block for a given block type.

Parameters:

ICODE Standard error code.

BN Data block name for the block to be defined.

BT Block type name; block-type definitions are created by way of the CDMS conversational interface DEFINE module.[5]

Example:

Suppose a user wishes to define the block JONES (to contain personal information about Tom Jones) for the block type PERSONAL (defined previously via the DEFINE module). The user might code

```
CALL DEFINB (ICODE, 5HJONES, 8HPERSONAL)
```

2. CHANGE

CHANGE (ICODE, BN, SBN, EN, DATA)

CHANGE adds or changes the data value(s) for either a single-valued element or an array element; in either case, the element must not be a member of a repeating group.

Parameters:

ICODE Both an input and an output parameter. If a single-valued element is to be changed, then ICODE is ignored as an input parameter. If an entire array is to be changed, then ICODE must

be set to zero; if a specific value in an array is to be changed, ICODE must be non-zero. As an output parameter, ICODE is a standard error code.

BN — Data block name

SBN — Subblock name or number

EN — Data element name

DATA — The value(s) to be added or inserted for the element EN. If EN is a single-valued element, then DATA simply contains the new value. If EN is an array and if a specific value of EN is to be changed, then DATA is a two-word array--the first word containing the subscript of the location within EN that is to be changed, and the second word containing the new value. If an array of new values is to be added or inserted for the array EN, the first word of DATA must contain the number of values in the new array (i.e., the dimension) and subsequent words will contain the values.

Example:

Suppose a data block JONES (a block containing personal data about Tom Jones) is of block type PERSONAL and contains the two subblocks SELF and FAMILY. Suppose further that data elements HOBBIES (an array) and PHONE (home telephone number; a single-valued element) are included in subblock SELF, and that the element CHILDREN is included in subblock FAMILY. Assume that none of these elements are members of a repeating group and

that the data base must be updated to account for the following:

(a) MR. Jones has a different home phone number, 793-5971;

(b) MR. Jones third hobby should read GOLF instead of GULF; and

(c) The Jones family now includes two children, DAVID and SUSAN.

To effect these changes, the user might code as follows:

```
      .
      .
      .
      INTEGER DATA (3)
      .
      .
      .
C SET UP PHONE NUMBER CHANGE
      BN = 5HJONES
      SBN = 4HSELF
      EN = 5HPHONE
      NUMBER = 8H793-5971
      CALL CHANGE (ICODE, BN, SBN, EN, NUMBER)
      IF (ICODE.NE.0) STOP 10
C SET UP HOBBY CHANGE
      ICODE = 1
      EN = 7HHOBBIES
      DATA(1) = 3
      DATA(2) = 4HGOLF
      CALL CHANGE (ICODE, BN, SBN, EN, DATA)
      IF (ICODE.NE.0) STOP 20
C SET UP CHILDREN CHANGE
```

```
SBN = 6HFAMILY
EN = 8HCHILDREN
DATA(1) = 2
DATA(2) = 5HDAVID
DATA(3) = 5HSUSAN
ICODE = 0
CALL CHANGE (ICODE, BN, SBN, EN, DATA)
IF (ICODE.NE.0) STOP 30
.
.
.
```

Note that the user could have saved several lines of code by entering BN, SBN, and EN as constants in the calling sequences.

3. ADDO

ADDO (ICODA, BN, SBN, RGN, DATA, LENGTH)

ADDO adds one occurrence to a particular repeating group. The new occurrence is always added at the end, thus becoming the last occurrence of the repeating group.

Parameters:

ICODA An array of error codes, one array location for each repeating group member whose name appears in the DATA array. If each location of ICODA is zero on return from ADDO, then the data values (contained in DATA) will have been successfully added as the last occurrence for RGN. If location ICODA(i) is set to 6 by ADDO, the ith element could not be processed and no new occurrence

will have been added to repeating group RGN. Other errors occurring in ADDO will cause ICODA(1) to be set to one of the other standard error codes.

BN — Data block name

SBN — Subblock name or number

RGN — Name of the repeating group to which the occurrence is to be added.

DATA — An array comprising consecutive entries of two or more computer words each, one entry for each RGN member that is to be processed by the current call to ADDO. The first word of each entry must be the member name (H format, left-justified). For a single-valued element, the second word of the entry must be the data value. For an array element, the second word must be the dimension, n, of the array; the next n words in the entry must be the array values.

LENGTH — The total number of words in DATA to be processed. The user should dimension DATA for at least LENGTH locations.

Example:

Suppose a block JONES (a block containing personal data about Tom Jones) is of block type PERSONAL and contains the elements TITLE (a single-valued element), SALARY (an array of salary information), and COWORKER (an array of pointers indicating persons who have worked with Tom Jones). Suppose further that these three elements are members of the repeating

group JOBS, and that the data base must now be updated to account for the fact that Mr. Jones has just begun his third job. To effect this change the user might code as follows:

```
      .
      .
      .
      DIMENSION ICODA (3), IDATA (12), DATA (12)
      EQUIVALENCE (DATA, IDATA)
      .
      .
      .
C SET UP CONTENTS OF ARRAY
C SET UP TITLE ENTRY
      DATA (1) = 5HTITLE
      DATA (2) = 7HCHEMIST
C SET UP SALARY ENTRY
      DATA (3) = 6HSALARY
      IDATA (4) = 1
      DATA (5) = 12000.0
C SET UP COWORKER ENTRY
      DATA (6) = 8HCOWORKER
      IDATA (7) = 5
      DATA (8) = 5HBROWN
      DATA (9) = 8HGOLDBERG
      DATA (10) = 5HADAMS
      DATA (11) = 6HNEWTON
      DATA (12) = 6HNELSON
      CALL ADDO (ICODA, 5HJONES, 1, 4HJOBS, DATA, 12)
      .
      .
      .
```

4. CHANGO

CHANGO (ICODA, BN, SBN, RGN, RGO, DATA, LENGTH)

CHANGO changes one occurrence of a specific repeating group. The member elements to be changed may be either single-valued elements or arrays. CHANGO may be used to add or replace a set of array values, or to replace a specific value in an array.

Parameters:

ICODA An array of error codes, one array location for each repeating group member whose name appears in the DATA array. As an output parameter, if each location of ICODA is set to zero, then all data values (contained in DATA) will have been successfully inserted into occurrence RGO. If location ICODA(i) is set to six by CHANGO, then the ith element could not be processed and no values will have been inserted into occurrence RGO for any element. If ICODA(i) is set to seven by CHANGO, the value for the ith element will not have been inserted into occurrence RGO. Other CHANGO errors will cause ICODA(1) to be set to one of the other standard error codes. As an input parameter, ICODA signifies the type of processing for arrays. If the ith element named in DATA is an array, location ICODA(i) will be examined. If ICODA(i) equals zero, CHANGO assumes that all current array values are to be replaced,

and that a new set of values is to be inserted into the occurrence. A non-zero ICODA(i) indicates that a specific array value is to be replaced.

BN — Data block name

SBN — Subblock name or number

RGN — Repeating group name

RGO — Repeating group occurrence number (a positive integer)

DATA — An array comprising consecutive entries of two or more computer words each, one entry for each RGN member that is to be processed by the current call to CHANGO. The first word of each entry must be the member name (H format, left-justified). For a single-valued element, the second word in the entry must be the data value. For an array element, the second word must be either the dimension of the new array, or the subscript of the specific value that is to be replaced in the existing array; the next word(s) in the entry must be the array value(s).

LENGTH — The total number of words in DATA to be processed. The user should dimension DATA for at least LENGTH locations.

Example:

Suppose a block JONES (a block containing personal data about Tom Jones) is of block type PERSONAL and contains the elements

TITLE (a single-valued element), SALARY (an array of real values), and COWORKER (an array of pointers). Suppose further that the three elements are members of repeating group JOBS, that Mr. Jones' position title in occurrence 4 of JOBS is spelled CHMEIST instead of CHEMIST, and that Mr. Jones' salary history and co-worker roster must be updated with the values 12,000.0, 12,850.0, and SMITH. To effect these changes the user might code as follows:

```
      .
      .
      .
      DIMENSION DATA (10), IDATA(10), ICODA(3)
      EQUIVALENCE (DATA, IDATA)
      .
      .
      .
C SET UP CONTENTS OF DATA
C SET UP TITLE ENTRY
      DATA (1) = 5HTITLE
      DATA (2) = 7HCHEMIST
C SET UP SALARY ENTRY
      DATA (3) = 6HSALARY
      DATA (4) = 2
      DATA (5) = 12000.0
      DATA (6) = 12850.0
C SET UP COWORKER ENTRY
      DATA (7) = 8HCOWORKER
C SET UP SUBSCRIPT FOR LOCATION WITHIN COWORKER
C THAT IS TO BE REPLACED
```

```
      IDATA (8) =3
      DATA (9) = 5HSMITH
C SET UP OTHER VALUES AS REQUIRED
      ISBN = 1
      ICODA (2) = 0
      ICODA (3) = 1
      CALL CHANGO (ICODA, 5HJONES, ISBN, 4HJOBS, 4, DATA, 9)
      .
      .
      .
```

5. ADDU

ADDU (ICODE, BN, SBN, EN, IV, DATA)

ADDU allows a user to add a locally defined element to a data block, and to also add the value(s) for that element.

Parameters:

ICODE As an input parameter, ICODE indicates the data type and array status of the new element, and must be equal to one of the following:

0	Alpha single-valued element
1	Integer single-valued element
2	Real single-valued element
3	Pointer single-valued element
10	Alpha array element
11	Integer array element
12	Real array element
13	Pointer array element
14	Text element

As an output parameter, ICODE is a standard error code.

BN — Data block name

SBN — Subblock name or number

EN — Data element name; must be unique for this data block

IV — Inversion indicator:

= 0 Consider the element to be non-inverted

≠ 0 Consider the element to be inverted

DATA — The value(s) to be added for element EN. If EN is a single-valued element, DATA will simply contain the value. If EN is an array, the first word of DATA will contain the number of values in the array while subsequent words will contain the values.

Example:

Suppose an employee named Tom Jones has a unique medical history. In the event of an emergency, it is necessary that Mr. Jones' personnel record (in this example, a data block named JONES) contain several vital pieces of information. To effect insertion of values for the elements BLOODTYP (a single-valued element) and ALLEGERY (an array), the user might code as follows:

```
.
.
.
EN = 8HBLOODTYP
DATA (1) = 6HAB NEG
```

```
ICODE = 0
CALL ADDU (ICODE, 5HJONES, 1, EN, 0, DATA)
IF (ICODE.NE.0) STOP 10
EN = 7HALLERGY
DATA(1) = 3
DATA(2) = 10HPENICILLIN
DATA(3) = 7HASPIRIN
DATA(4) = 9HNOVACAINE
ICODE = 10
CALL ADDU (ICODE,5HJONES, 1, EN, 0, DATA)
IF (ICODE.NE.0) STOP 20
.
.
.
```

6. DELETE

DELETE (ICODE, BN, SBN, RGN, RGO, EN)

DELETE deletes elements (either single-valued or array) a data block. The subroutine can delete a non-repeating group element, or it can delete one or all occurrences of a repeating group element. If required, one occurrence or all occurrences of a repeating group can be deleted.

Parameters:

ICODE	Standard error code
BN	Data block name
SBN	Subblock name or number
RGN	Repeating group name; used only when a specific occurrence or all occurrences are to be deleted

RGO Repeating group occurrence number (ignored when a non-repeating group element is to be deleted):

= 0 Process all occurrences

= i, where i > 0 Process ith occurrence

EN When one or all occurrences of an entire repeating group are to be deleted, EN must be set to zero. Otherwise, it is used to indicate the name of the data element to be deleted.

Examples:

Use of parameters RGN, RGO, and EN is summarized in the following table and calling sequences:

DELETE Processing Actions	Parameter Value		
	RGN	RGO	EN
1. all occurrences of a repeating group	repeating group name	0	0
2. ith occurrence of a repeating group	repeating group name	i	0
3. all occurrences of a repeating group element	ignored	0	element name
4. ith occurrence of a repeating group element	ignored	i	element name
5. non-repeating group element	ignored	ignored	element name

(1) CALL DELETE (ICODE, 5HJONES, 8HINDIVIDL, 0, 0, 6HWEIGHT)

(2) CALL DELETE (ICODE, 5HJONES, 8HINDIVIDL, 4HJOBS, 0, 0)

(3) CALL DELETE (ICODE, 5HJONES, 8HINDIVIDL, 0, 2, 4HYEAR)

(4) CALL DELETE (ICODE, 5HJONES, 8HINDIVIDL, 4HJOBS, 2, 0)

(5) CALL DELETE (ICODE, 5HJONES, 8HINDIVIDL, 0, 0, 4HYEAR)

In (1), DELETE is called to delete the value for the element WEIGHT from the block JONES (assume WEIGHT to be a non-repeating group element). In (2), all occurrences of the repeating group JOBS are to be deleted. In (3), the element YEAR is to be deleted from occurrence 2 of JOBS (assume YEAR to be a member of JOBS). In (4), the entire second occurrence of JOBS is to be deleted. Finally, in (5), the element YEAR is to be deleted from all occurrences of the repeating group JOBS.

7. DELETB

DELETB(ICODE, BN)

DELETB deletes a data block. Further reference to the block is thus impossible.

Parameters:

ICODE Standard error code

BN Data block name

Example:

To delete a data block JONES, the user might code

CALL DELETB (ICODE, 5HJONES)

8. MODBN

MODBN (ICODE, OLDBN, NEWBN)

MODBN allows the user to change the name of a data block.

Parameters:

ICODE Standard error code

OLDBN Data block name to be changed

NEWBN New data block name

9. PFNIN

PFNIN (ICODE, DATA, PFN, ID, FILTYP)

PFNIN stores the name of the permanent file referenced by a cross-file pointer, and then completes the definition of that pointer by inserting a cross-file reference table index indicating where the file name has been stored. The index is used by subroutine INPFN during cross-file pointer processing.

Parameters:

ICODE — Standard error code

DATA — A variable containing a pointer of as many as eight characters, left-justified, H format. Upon completion of PFNIN, DATA will also contain a cross-file reference table index in the 12 right-most bit positions.

PFN — Name of the permanent file containing the data block identified by the pointer in DATA. If a SCOPE file, PFN must contain the permanent file name of as many as 40 characters. If a CPFMS file, PFN must contain the level-3 name of as many as eight characters, left-justified, H format.

ID — For a SCOPE file, the four-character ID; for a CPFMS file, the level-4 name of as many as eight characters, left-justified, H format.

FILTYP — File type code:

= 0 SCOPE file

= 1 CPFMS file

E. DATA BLOCK RETRIEVAL SUBROUTINES

1. FETCH

FETCH (ICODE, BN, SBN, RGN, RGO, EN, DATA, LENGTH)

FETCH can retrieve any data element (either single-valued or array) whether or not the element is a member of a repeating group. If in a repeating group, either a specific occurrence of the element or all occurrences of the element may be retrieved. Moreover, one occurrence or all occurrences of a repeating group may be retrieved.

Parameters:

ICODE	Standard error code
BN	Data block name
SBN	Subblock name or number
RGN	Repeating group name; used only when a specific occurrence or all occurrences are to be retrieved
RGO	As an input parameter, a repeating group occurrence number: = 0 Process all occurrences = i where i > 0 Process ith occurrence As an output parameter, the total number of occurrences in the repeating group. RGO is ignored when a non-repeating group element is to be retrieved.
EN	When one occurrence or all occurrences of an entire repeating group are to be retrieved, EN must be set to zero. Otherwise, the name of the data element to be retrieved.

DATA A user-supplied work space used to store the value(s) retrieved by FETCH. The structure of DATA is dependent upon the type of processing action performed by FETCH. The following lists summarize FETCH processing actions and potential organizations of the DATA array.

FETCH Processing Actions:

(1) A non-repeating group single-valued element or a specific occurrence of a single-valued repeating group element

(2) A non-repeating array element or a specific occurrence of a repeating group array element

(3) All occurrences of a repeating group element

(4) One occurrence of a repeating group

(5) All occurrences of a repeating group

DATA organizations (keyed to processing actions):

(1) Contains the value of the single-valued element

(2) Contains the dimension, n, of the array in the first word and the array values in the subsequent n words

(3) Contains consecutive entries, one for each repeating group occurrence. If the element is single-valued, each entry will contain either (a) a value, or (b) an octal pattern of 7676767676767676 indicating that no value has been loaded for this occurrence. If the element is an array, each entry will

contain either (a) a dimension word followed by the array values, or (b) a zero indicating that no values have been loaded for this occurrence.

(4) Contains consecutive entries, one for each member of the repeating group. The entries are ordered in the same way that the repeating group elements are defined in the block-type definition. The entry for a single-valued element will be either a value, or an octal pattern of 7676767676767676767 signifying no value has been loaded for this occurrence. The entry for an array will be either a dimension word followed by the array values, or a zero indicating no values have been loaded for the occurrence.

(5) Contains sets of consecutive entries as described in DATA organization (4). The sets themselves are consecutive and are sequenced according to the repeating group occurrence number.

LENGTH As an input parameter, the total number of words in DATA that can be used for retrieved information; the user must dimension DATA to have at least LENGTH words. As an output parameter, the number of words in DATA which actually contain retrieved information.

Example:

A user might code

```
.
.
.
DIMENSION DATA (50)
.
.
.
LENGTH = 50
IRGO = 0
CALL FETCH (ICODE, 5HJONES, 6HFAMILY, 8HMARRIAGE, IRGO, 0,
  DATA, LENGTH)
.
.
.
```

Execution of this code results in the retrieval of all occurrences of the repeating group MARRIAGE from the block JONES. If the block-type definition for MARRIAGE comprises the elements WIFE (a single-valued element), YEARM (also a single-valued element), and CHILDREN (an array), then DATA will appear as follows:

Word	
1	ELLEN
2	1885
3	3
4	MARGARET
5	SUSAN
6	ERIN
7	CAROLYN
8	1915
9	0

Two occurrences, representing marriages to Ellen and Carolyn, have been retrieved; the second marriage resulted in no children as indicated by the zero. The parameter LENGTH would contain the value 9.

2. FETCHN

FETCHN (ICODA, BN, SBN, ENA, DATA, LENGTH)

FETCHN retrieves data values for a set of non-repeating group elements; the set may consist of single-valued elements and array elements.

Parameters:

ICODA — An array of error codes, one array location for each data element named in the ENA array. If ICODA (i) is set to 3 by FETCHN, no value has been loaded for element ENA(i). If ICODA(i) is set to 6 by FETCHN, the element ENA(i) is not a valid element for the block EN. In either case, the data values for all other elements named in ENA (and which have a zero in the appropriate locations of ICODA) will be retrieved and placed into the DATA array. Other error conditions detected by FETCHN will cause ICODA(1) to be set to the appropriate standard error code.

BN — Data block name

SBN — Subblock name or number

ENA — An array containing the names of one or more non-repeating group elements having values to be retrieved by FETCHN. The array must contain only one name per location (H format, left-justified); the last location of ENA must be set to zero.

DATA — An array composed of consecutive entries, one

for each element named in ENA having its corresponding location in ICODA a zero. The entry for a single-valued element will contain simply the element value. The entry for an array element will include a dimension word followed by the array values.

LENGTH As an input parameter, the total number of words in DATA that can be used for retrieved information; the user should dimension DATA to have at least LENGTH words. As an output parameter, the number of words in DATA actually containing retrieved information.

3. FETCHR

FETCHR (ICODA, BN, SBN, RGN, RGO, ENA, DATA, LENGTH)

FETCHR retrieves data values for a set of repeating group elements; the set may consist of single-valued elements and array elements. Values may be retrieved for either a specific occurrence or all occurrences of a repeating group.

Parameters:

ICODA An array of error codes, one location for each repeating group element named in the ENA array. If ICODA(i) is set to 3 by FETCHR, the element ENA(i) has no value loaded for the occurrence(s) requested. If ICODA(i) is set to 6 by FETCHR, the element ENA(i) is not a valid element for block BN. Other error conditions will cause ICODA(1)

to be set to the appropriate standard error code.

BN Data block name

SBN Subblock name or number

RGN Repeating group name; each element named in the ENA array must be a member of RGN

RGO Repeating group occurrence number:

= 0 Retrieve all occurrences

= i, where i>0 Retrieve ith occurrence

Whenever all occurrences are to be retrieved, RGO will be an output parameter indicating the number of occurrences actually processed.

ENA An array containing the names of one or more repeating group elements having values to be retrieved. The array must contain only one name per location (H format, left-justified); the last location of ENA must be zero.

DATA An array composed of consecutive entries, one for each element named in ENA having a value other than 6 in the corresponding ICODA location; the entries are made in the same order as were the ENA names. Whenever all occurrences of a repeating group are to be retrieved, all entries for the first occurrence will be first, all entries for the second occurrence next, etc. The entry for a single-valued element will be either a value, or an octal pattern of 7676767676767676767676 signifying no value has been loaded for this occurrence. The entry for an array

will be either a dimension word followed by the array values, or a zero indicating no values have been loaded.

LENGTH An an input parameter, the total number of words in DATA that can be used for retrieved information; the user should dimension DATA for at least LENGTH words. As an output parameter, LENGTH is the number of words in DATA that contain retrieved information.

4. QUERY

QUERY (ICODE, LUN, LENGTH, PARM, HITTYP)

Recall that an inverted list is a directory which contains (a) the data values for a particular data element given inversion status when it was defined, and (b) the names of all blocks having values for the inverted elements. QUERY enables the user's program to search inverted lists for the names of those data blocks that satisfy certain user-specified criteria. The following criteria may be provided:

(i) A relational expression, that is, an expression either of the form

element name operator value

where: element name — Name of inverted element

operator — .EQ.,.GE.,.GT.,.LE.,.LT., or .NE.

value — Data value to which the element's value is compared

or of the form

element name .BET. value 1, value 2

where: element name — Same as above

.BET. — Operator indicating that only values within the range defined by value 1 and value 2 inclusive will be considered

value 1, value 2 — Data values with which the element's value is compared

(ii) A block-type expression, that is, an expression of the form:

BT.EQ.type

where: type — Block type name

(iii) A combination of up to five relational and block-type expressions joined by the Boolean operators .AND. and .OR., for example,

DEPTH.EQ.10.OR.HEIGHT.LT.5.AND.WIDTH.BET.20,30

If blocks X, Y, and Z have element DEPTH equal to 10, and if blocks A, B, C, and D have element HEIGHT less than 5, and if blocks C, D, G, and H have element WIDTH between 20 and 30 inclusive, then the block names C, D, X, Y and Z will be returned to the user's program. Parentheses may be used to alter the normal .AND. over .OR. precedence.

Parameters:

ICODE — Standard error code

LUN — Logical unit number (in the range 1-99 inclusive) of a "hit" file on which the names of those blocks

that satisfy the user's retrieval criteria will be written. The file is organized as a set of 128-word binary records.

LENGTH As an input parameter, the length of the PARM array (in the range 3-30 inclusive). As an output parameter, the number of block names written on the "hit" file.

PARM An array containing the user's retrieval criteria, one array location for each item expressed in the criteria. Retrieval items include parentheses, element names, relational and Boolean operators, and data values. Items may be specified in various ways:

(a) By parenthesis, 1H(or 1H)

(b) By element name. The element name must be H format, left-justified. Moreover, the two rightmost bits of the location containing the element name must indicate the data type according to the convention:

0 real

1 integer

2 alphanumeric

3 pointer

(c) By relational operator, indicated by the integer code:

1 .BET.

2 .EQ.

3 .GE.

4 .GT.

5 .LE.

6 .LT.

7 .NE.

(d) By Boolean operator, indicated by the integer code:

9 .AND.

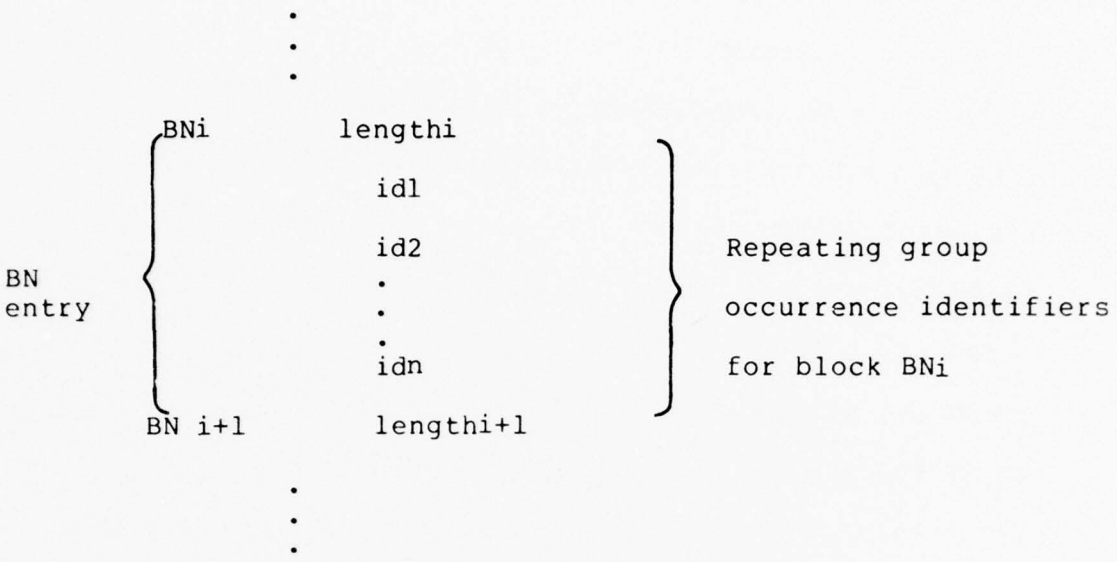
10 .OR.

(e) By values. Values of the type integer or real must be provided in FORTRAN fixed and floating formats, respectively. Values of the type alphanumeric or pointer must be provided in H format, left-justified.

HITTYP Information type code:

= 0 Write only data block names (one name per word, H format, left-justified) to the "hit" file

≠ 0 Write block names and repeating group occurrence identifiers to the "hit" file. (A repeating group occurrence identifier relates an inverted list value with the particular repeating group occurrence which contains that value. The identifier is useful only in a call to subroutine FETCHC and must not be confused with the repeating group occurrence number so frequently used by the CDMS host language subroutines.) The "hit" file is structured in the following manner:



The parameter lengthi is a 12-bit quantity equal to the number of id's plus one; an id is simply an integer value. More than one BN entry may be present in the 128-word "hit" file output record: The last complete entry will be followed by a zero word; the next entry (if one exists) will begin in the first word of the next output record.

Example:

A user desiring to know which of the United States Presidents born between 1800 and 1899 were members of the Republican Party might formulate the retrieval criteria

YEARB.BET. 1800, 1899 .AND. PARTY .EQ. REPUBLICAN

and might code

```
.
.
.
DIMENSION IPARM (8)
.
.
.
```

```
C SET UP IPARM ARRAY ACCORDING TO THE RETRIEVAL CRITERIA
      IPARM (1) = 5HYEARB
      IPARM (1) = OR (AND(IPARM(1), MASK(58)),1B)
C THE ABOVE STATEMENT PACKS THE DATA TYPE (INTEGER) FOR
C THE ELEMENT YEARB
      IPARM (2) = 1
      IPARM (3) = 1800
      IPARM (4) = 1899
      IPARM (5) = 9
      IPARM (6) = 5HPARTY
      IPARM (6) = OR (AND(IPARM(6), MASK (58)), 2B)
      IPARM (7) = 2
      IPARM (8) = 10HREPUBLICAN
      LENGTH = 8
      CALL QUERY (ICODE, 35, LENGTH, IPARM, 0)
      IF (ICODE.NE.0) STOP 10
C THE FILE TAPE35 WILL NOW CONTAIN THE NAMES OF
C THE PRESIDENTS THAT SATISFY THE RETRIEVAL CRITERIA
      .
      .
      .
```

5. FETCHC

FETCHC(ICODE, BN, SBN, RGN, RGID, EN, DATA, LENGTH)

FETCHC allows a user to retrieve one or more data values from a specific repeating group occurrence that satisfies certain user-specified retrieval criteria. FETCHC operates in association with subroutine QUERY (discussed previously).

Values may be retrieved for either a single repeating group element, or the entire repeating group occurrence.

Parameters:

ICODE	Standard error code
BN	Data block name
SBN	Subblock name or number
RGN	Repeating group name; used only when an entire occurrence is to be retrieved
RGID	As an input parameter, a repeating group occurrence identifier obtained by way of subroutine QUERY; the identifier marks the occurrence from which values will be retrieved. As an output parameter, the number of the repeating group occurrence containing the identifier which was provided as input.
EN	A value of zero if one occurrence of a repeating group is to be retrieved; otherwise, the name of the data element to be retrieved.
DATA	A user-supplied work space used to store the value(s) retrieved by FETCHC. If a single-valued element is to be retrieved, DATA will contain only the value for that element. If an array is to be retrieved, DATA will contain a dimension word followed by the array values. If an entire occurrence is to be retrieved, DATA will be composed of consecutive entries, one for each member of the repeating group. The entries will

be ordered in the same way as the repeating group elements are defined in the block type. The entry for a single-valued element will be either a value, or an octal pattern of 76767676767676767676 signifying no value has been loaded for the occurrence. The entry for an array will be either a dimension word followed by the array values, or a zero indicating no values have been loaded.

LENGTH As an input parameter, the total number of words in DATA that can be used for retrieved information; the user must dimension DATA to have at least LENGTH words. As an output parameter, the number of words in DATA which actually contain retrieved information.

6. NOCC

NOCC(ICODE, BN, SBN, RGN, RGO)

NOCC determines the number of repeating group occurrences that exist within a given data block.

Parameters:

ICODE Standard error code

BN Data block name

SBN Subblock name or number

RGN Name of the repeating group for which a count of the occurrences is to be determined

RGO Number of occurrences for RGN

7. INPFN

INPFN(ICODE, DATA, PFN, ID, FILTYP)

INPFN retrieves the name of the permanent file referenced by a cross-file pointer. An index, inserted into the pointer by subroutine PFNIN, indicates where in the cross-file reference table the file name is stored.

Parameter:

ICODE Standard error code

DATA A variable containing a pointer of as many as eight characters (left-justified, H format), and a cross-file reference table index (in the 12 right-most bit positions). If the index is zero, signifying a within-file pointer, PFN and ID will each contain 10 blanks on return from INPFN.

PFN Name of the file containing the block referenced by the value in DATA. If a SCOPE file, PFN will contain the permanent file name of as many as 40 characters. If a CPFMS file, PFN will contain the level-3 name of as many as eight characters, left-justified, H format.

ID For a SCOPE file, the four-character ID; for a CPFMS file, the level-4 name of as many as eight characters, left-justified, H format.

FILTYP File type code:

= 0 SCOPE file

= 1 CPFMS file

ACKNOWLEDGMENTS

The authors wish to acknowledge Thomas R. Rhodes of the David W. Taylor Naval Ship Research and Development Center (DTNSRDC) and Roger Berry of the Naval Ship Engineering Center (NAVSEC) for their technical suggestions and criticisms.

APPENDIX A

COMRADE PERMANENT FILE MANAGEMENT SYSTEM

The COMRADE Permanent File Management System (CPFMS) affords the COMRADE user/programmer a special file management capability otherwise not available under the SCOPE permanent file system. Using CPFMS, the user/programmer is able to limit access to only those persons associated with a particular design project. This control is achieved by creating CPFMS files which may only be accessed from within a COMRADE application system.

FILE-NAMING CONVENTION

Each CPFMS file is assigned a unique four-level designation. The L1 (first-level) name CPFM is given to all CPFMS files. The L2 (second-level) name is that of the application system with which the file is associated (for example, ISDS). The L3 (third-level) name is that of a project within the application system with which the file is associated (for example, DDG3). The L4 (fourth-level) name may be freely chosen; it is used to suggest the contents of the file (WEIGHTS, for example). The L2, L3, and L4 names are concatenated, with dashes, to form a unique permanent file name which can be processed by the SCOPE permanent file system. Each level name many be composed of as many as eight characters. The designation ISDS-DDG3-USERS (with ID=CPFM) typifies the structure of a CPFMS file name.

FILE ACCESS CONTROL

CPFMS files may be used only within the application system for which they were created; thus only the L3 and L4 names actually need be submitted by the would-be user. Two schemes for controlling access to CPFMS files from within application systems are provided--the pseudo-password, and the File-Access-Key/File-Access-Lock (FAK/FAL). Both are assigned at the time the file is cataloged within the CPFMS system. If the pseudo-password is used, access to that file will require that the user submit a pseudo-password for a check. If the password submitted by the user is identical to the one assigned at the time the file was cataloged, access is granted.

If the FAK/FAL is used, the operation is more complex. The FAL assigned to a file consists of a 20-bit field, each bit associated with a different aspect of a project effort (for example, the hull design portion). The FAK assigned to each COMRADE user consists of 60 bits, organized into three 20-bit fields called sub-FAK's. Each of the three sub-FAK's pertains to a different level of file access: (1) Attach with read permission only; (2) Attach with read-write permission; or (3) Attach with purge permission. Each bit of the 20-bit sub-FAK may itself be associated with a particular subset of the project effort. Those sub-FAK bit positions that correspond to the subsets that the user is allowed to use will contain 1's. Each of the 20-bit sub-FAK's will be compared with the FAL of the file in question to determine which, if any, of

these levels of file-access will be permitted that particular user. The FAK/FAL check will be considered a success (that is, access will be granted) if the bits in the user's 20-bit sub-FAK that correspond to the 1-bits contained in the FAL are 1's.

APPENDIX B

DTNSRDC CHARACTER SET

DISPLAY CODE	CHARACTER	PUNCH 026	PUNCH 029 IF DIFF	7-TRACK TAPE	NOTE NAME
01	AAA	12-1		61	
02	BBB	12-2		62	
03	CCC	12-3		63	
04	DDD	12-4		64	
05	EEE	12-5		65	
06	FFF	12-6		66	
07	GGG	12-7		67	
10	HHH	12-8		70	
11	III	12-9		71	
12	JJJ	11-1		41	
13	KKK	11-2		42	
14	LLL	11-3		43	
15	MMM	11-4		44	
16	NNN	11-5		45	
17	OOO	11-6		46	
20	PPP	11-7		47	
21	QQQ	11-8		50	
22	RRR	11-9		51	
23	SSS	0-2		22	
24	TTT	0-3		23	
25	UUU	0-4		24	
26	VVV	0-5		25	
27	WWW	0-6		26	
30	XXX	0-7		27	
31	YYY	0-8		30	
32	ZZZ	0-9		31	
33	000	0		12	
34	111	1		01	
35	222	2		02	
36	333	3		03	
37	444	4		04	
40	555	5		05	
41	666	6		06	
42	777	7		07	
43	888	8		10	
44	999	9		11	
45	+++	12	12-8-6	60	PLUS
46	---	11		40	MINUS
47	***	11-8-4		54	ASTERISK
50	///	0-1		21	SLASH
51	(((	0-8-4	12-8-5	34	LEFT PAR
52	)))	12-8-4	11-8-5	74	RT PAR
53	$$$	11-8-3		53	DOLLAR

DISPLAY CODE	CHARACTER	PUNCH 026	PUNCH 029 IF DIFF	7-TRACK TAPE	NOTE NAME
54	===	8-3	8-6	13	EQUAL
55				20	BLANK
56	,,,	0-8-3		33	COMMA
57	...	12-8-3		73	PERIOD
60	###	0-8-6	8-3	36	POUND
61	[[[	8-7	12-8-2	17	L BRACKET
62	]]]	0-8-2	11-8-2	32	R BRACKET
63	:::	8-2			COLON
64	"""	8-4	8-7	14	QUOTE
65	___	0-8-5		35	UNDERLINE
66		11-8-2	12-8-7	52	EXCLAMATION
66		11-0		52	EXCLAMATION
67	&&&	0-8-7	12	37	AMPERSAND
70	'''	11-8-5	8-5	55	APOSTROPHE
71	???	11-8-6	0-8-7	56	QUESTION
72		12-8-2	12-8-4	72	LESS THAN
72		12-8		72	LESS THAN
73		11-8-7	0-8-6	57	GREATER
74	@@@	8-5	8-4	15	AT
75		12-8-5	0-8-2	75	REVERSE SLANT
76		12-8-6	11-8-7	76	CIRCUMFLEX
77	;;;	12-8-7	11-8-6	77	SEMICOLON
55		8-6	0-8-4		BLANK

APPENDIX C

SUMMARY OF CDMS STORAGE CAPACITIES

MAXIMUM DATA BLOCK STORAGE CAPACITIES:

Number of data blocks on each file	2**53-1
Number of words in each data block	262,143
Number of blocks that may be saved at any one time for reallocation	81,090
Number of full or partial data blocks that may be contained in buffer at any one time	20

MAXIMUM INVERTED LIST STORAGE CAPACITIES:

Number of inverted element lists per file	100
Number of entries per inverted list element	173,740
Number of qualifying inverted list entries per query	64,897

MAXIMUM BLOCK TYPE STORAGE CAPACITIES:

Number of block types on each file	100
Number of data elements per block type	250
Number of subblocks per block type	8
Number of repeating groups per subblock	14
Number of elements per repeating group	50

APPENDIX D

HOST LANGUAGE INTERFACE OPERATING PROCEDURES

A. SUBROUTINE ACCESS

The subroutines constituting the CDMS host language interface reside on a program library created by way of the SCOPE EDITLIB capability.[3] This library, a SCOPE permanent file on DTNSRDC's Control Data 6700 computer, is named SUBLIB, ID=COMR. All users are automatically assigned "read only" permission when they access SUBLIB for use in creating their application programs. SUBLIB contains five groups of subroutines:

- CDMS host language interface subroutines, both the user-callable and the more primitive subroutines.
- COMRADE utility subroutines (see Appendix E)
- Absolute Subroutine Utility (ASU) subroutines which are required for the access of the COMRADE Absolute Subroutine Library (see Appendix F)
- ALPHAPACK subroutines which are used for alphanumeric data conversion (see Appendix G)
- Permanent file management subroutines (Appendix H)

The control statements needed to access the host language interface subroutines for the purpose of creating an application program are

```
ATTACH, SUBLIB, ID = COMR.
LIBRARY, SUBLIB.
```

Thus, given that a FORTRAN program exists on a local file named HULLDEF, the following control statements would compile the

source code for HULLDEF and then load the object code including the host language subroutines, and would then execute the program:

```
FIN, I = HULLDEF.
ATTACH, SUBLIB, ID = COMR.
LIBRARY, SUBLIB.
LDSET, USEP = DBMF/CDSF/CDSFI.
LGO.
```

There are two variations to this procedure that are worth investigation by a CDMS user:

(1) Suppression of the Automatic Reprieve Feature. The standard use of SUBLIB (just described) provides for an automatic "reprieve" from abnormal terminations such as an address out-of-range, or the use of an infinite or indefinite operand in an arithmetic expression. The user who does not wish his program to be reprieved can circumvent the automatic "reprieve" by inserting the statement

```
LDSET, SUBST = OPEN-ZNOPEN.
```

preceding the LGO statement.

(2) Inverted List Bypass. The CDMS host language subroutine IFLN supports the option of by-passing the processing of any inverted list transactions. Use of this option could result in a core saving of approximately 7000 (octal) locations, since the subroutines and labelled common blocks required to process the transactions would not be needed. The user who does not wish his program to contain the inverted list processing software

should insert the statements

```
LDSET,SUBST=ADDIE-ZNADDIE/CHNGIE-ZNCHGIE/DELIE-ZNDELIE.
LDSET,SUBST=BRKUP-ZNBRKP/REFILE-ZNRFILE
```

before the LGO statement, and should remove the designation /CDFSI from the LDSET,USEP statement. The inverted list processing software is used by many of the host language update subroutines; only one retrieval subroutine, QUERY, accesses inverted lists.

B. CORE REQUIREMENTS

Of the approximately 250 subroutines on SUBLIB, 60-70 are user-callable. Since these subroutines are not mutually exclusive (that is, subroutines A, B, and C might each call subroutine X, but subroutine A might also call subroutine Y; subroutine B might also call subroutine Z; while subroutine C might call Y and Z), computation of the core requirements of a particular application program is difficult. One way of estimating these core requirements is by constructing a dummy program that calls the needed SUBLIB subroutines, and then allowing the system loader to compute the storage needs. Thus, a user desiring to estimate the core required for subroutines SBUFF, ILFN, FETCH, CHANGE, ADDO, and FLFN might code

```
PROGRAM TEST
CALL SBUFF
CALL ILFN
CALL FETCH
CALL CHANGE
CALL ADDO
CALL FLFN
END
```

The control statements needed to effect loading of such a program are

```
FTN.
MAP, ON.
ATTACH, SUBLIB, ID = COMR.
LIBRARY, SUBLIB.
LOAD, LGO.
NOGO.
```

Following inspection of the system loader's computations, the user could determine the need for a sophisticated program structuring facility such as overlay[3], segmentation[3], or the COMRADE Absolute Subroutine Utility[9] (see Appendix F).

APPENDIX E

COMRADE UTILITY SUBROUTINES

The library SUBLIB contains a set of utility subroutines which have been found to be quite useful to DTNSRDC Control Data 6700 programmers. The subroutines allow an application program to

- pack a value into a user-specified field within a given word (PACK),
- unpack a value from a user-specified field within a given word (UNPACK),
- change blank characters to binary zeros (FILL),
- determine whether the calling program is executing in the conversational or batch mode (RWE),
- compress blank characters from a card image (RPK), and
- terminate the user program (TERMJOB).

Individual descriptions of the utility subroutines follow.

1. PACK (TO, IL IR, FROM, ICODE)

PACK packs a value into a user-specified field within a given computer word.

Parameters:

TO	A word into which a value will be packed
IL, IR	The leftmost and rightmost bit positions, respectively, of the field within TO into which a value will be packed. According to convention, the bits of a Control Data 6700 word are numbered right to left, 0 through 59; thus 59 $\geq$ IL $\geq$ IR $\geq$ 0.

FROM Value to be packed in TO

ICODE Error code:

= 0 No error detected

= 1 IL $\geq$ 59

= 2 IR $\leq$ 0

= 3 IR $\geq$ IL

= 4 Value too large for the field specified

Example:

Suppose that the variables I and J have the following bit patterns:

	59	58	57	...	38	37	36	35	...	3	2	1	0
I	1	1	1	...	1	1	1	1	...	1	1	1	1

	59	58	57	...	38	37	36	35	...	3	2	1	0
J	0	0	0	...	0	0	0	0	...	1	0	0	1

After the statement

CALL PACK (I, 38, 35, J, ICODE)

has been executed, I will have the contents

59	58	57	...	38	37	36	35	...	3	2	1	0
1	1	1	...	1	0	0	1	...	1	1	1	1

and J will be unchanged.

2. UNPACK

UNPACK (FROM, IL, IR, TO, ICODE)

UNPACK unpacks a value from a user-specified field within a given computer word.

Parameters:

FROM A word from which a value will be unpacked.

IL, IR The leftmost and rightmost bit positions, respectively, of the field within FROM from which a value will be unpacked. The bit positions of a Control Data 6700 word are numbered right to left, 0 through 59; thus $59 \geq IL > IR \geq 0$.

TO A word into which the unpacked value will be placed, right-justified.

ICODE Error code:

= 0 No error detected

= 1 $IL \geq 59$

= 2 $IR \leq 0$

= 3 $IR \geq IL$

Example:

Suppose the variable I has the following bit pattern

59	58	57	56	55		0
1	0	0	1	1	...	1

After the statement

```
CALL UNPACK (I, 59, 56, J, ICODE)
```

has been executed, J will have the contents

59				3	2	1	0
0	...	0	1	0	0	1	

and I will be unchanged.

3. FILL

```
FILL (NAME, ICODE)
```

FILL changes blank characters (display code 55 octal) in a given computer word to binary zeros (display code 00) and vice versa.

Parameters:

NAME Location containing the characters to be changed.

ICODE Change code:

= 1 Zeros to blanks

= 2 Blanks to zeros

4. RWE

RWE (MODE)

RWE determines whether the application program is executing in conversational or batch mode.

Parameters:

MODE Operation code:

= 0 Batch mode

= 1 Conversational mode

5. RPK

RPK (STRING, LAST)

RPK compresses blank characters from an 80-character card image.

Parameters:

STRING As an input parameter, an 8-word array containing the card image to be compressed. As an output parameter, a set of non-blank characters, left-justified, H format.

LAST Number of non-blank characters in STRING.

6. TERMJOB

TERMJOB

TERMJOB terminates the application program, producing no day-file messages and eliminating the need for a FORTRAN STOP statement.

APPENDIX F

USE OF THE COMRADE ABSOLUTE SUBROUTINE LIBRARY

A. SUBROUTINE ACCESS

The COMRADE Absolute Subroutine Utility (ASU)[9] provides a capability for loading and executing the CDMS host language subroutines as well as those auxiliary subroutines identified in Appendix D. Two ASU subroutines - INITLDR and COMRLDR - are used to load and execute CDMS subroutines from an Absolute Subroutine Library (ASL). Use of an ASL offers the following benefits:

- The amount of core required to run an application program can be reduced, since the CDMS host language subroutines (and the auxiliary software) can execute within the same core area.
- Only one copy, the latest, of the CDMS subroutines is needed, since the subroutines reside on an ASL that is designed to be shared by many application programs.
- Since ASL subroutines are separate from an application program, subroutine errors can be corrected without requiring that users recreate their programs.
- Parameters may be passed to and among ASL subroutines either as arguments in the CALL to COMRLDR, or through a shared common block.
- No instruction modification is needed since ASL subroutines are in absolute form.

Two permanent files are required when the COMRADE ASL subroutines are to be used. The files are named ASL, ID = COMR and SUBLIB, ID = COMR. The file named ASL contains COMRADE subroutines in absolute form and must be attached whenever an application program is to be executed. The file named SUBLIB (discussed in Appendix D) contains the subroutines INITLDR and COMRLDR and is required when the application program is being created. Descriptions of the two subroutines follow.

1. INITLDR

INITLDR (LFN, NAME, IERR)

INITLDR identifies the ASL file that contains the COMRADE subroutines in absolute form. INITLDR should be called only once in an application program, before the first CALL to COMRLDR.

Parameters:

LFN — Logical file name (left-justified, H format) of the ASL file.

NAME — Reprieve indicator:

= 0 No reprieve subroutine is to be executed

= REFILE (left-justified, H format) A reprieve subroutine (developed by COMRADE system personnel) named REFILE will be executed following an abnormal error condition; REFILE closes and finalizes the CDMS data base file active at the time of the error. If a user-developed subroutine is to be executed, the name of that subroutine, instead of REFILE, must appear in NAME.

IEER Error code

= 0 No error detected

≠ 0 File not in ASL format

2. COMRLDR

COMRLDR (SUB, PARM1, PARM2,...PARMN)

COMRLDR loads and executes a specific ASL subroutine. If the subroutine is already in memory, execution will proceed directly. If it is not resident on the current ASL, the message INVALID SUBROUTINE NAME will be written to the dayfile and the program will be aborted. If subroutine INITLDR has not been called, the message INITLDR NOT CALLED will be written to the dayfile and the program aborted.

Parameters:

SUB The name (left-justified, H format) of the ASL subroutine that is to be executed.

PARM1- PARMN The parameters to be passed to the subroutine identified by SUB; the list is variable-length.

Example:

```
CALL COMRLDR (4HILFN, IERR, 3HSDF, 0, 1)
CALL COMRLDR (5HFETCH, ICODE, BN, SBN, RGN, RGO, EN, DATA,
  LENGTH)
```

B. CORE REQUIREMENTS

Before ASL subroutines can be used, a core area in which the subroutines will execute must have been defined. This area must coincide with the area in which the subroutines have been designed to execute, otherwise an error will occur. The COMRADE

ASL subroutines have been designed to execute in an area that begins at location 101 (octal) and will require a maximum of 20K (octal) words. A smaller area may be provided, depending on the specific subroutine to be called. Once the size of the area has been determined, it can be reserved by way of a FORTRAN labelled common statement placed immediately after the PROGRAM statement. The statement may appear as

```
COMMON/AREA/DUM (20000B)
```

A list of subroutines resident on the file ASL, ID = COMR follows. The core requirement for each is indicated.

COMRADE ASL SUBROUTINES

NAME	LENGTH (OCTAL)	NAME	LENGTH (OCTAL)
ADDO	17101	FETCHC	12633
ADDU	16770	FETCHN	12515
ATTACH	7315	FETCHR	13503
BLKLEN	7276	FLFN	15155
BRKUP	12010	FTNBIN	12355
CAD	17131	GET	10221
CAI	17076	GETCOM	7107
CAO	17070	ILFN	10713
CAR	17114	INPFN	6254
CATLOG	7315	LOCATE	6322
CDA	17204	LSHIFT	6213
CHANGE	17725	MODBN	15722
CHANGO	17375	MOVE	6276
CIA	17277	NOCC	12266
CLUNLD	11516	PACK	6250
CLRWND	11516	PASOVR	7756
COA	17277	PERM	7120
COMATC	15271	PFNIN	6644
COMCAT	15540	PURGE	7315
COMPRG	15320	QUERY	17524
COMUNL	13535	REFILE	12304
CRA	17202	RENAME	7315
CULL	13727	REQUEST	6422
DEFFIL	7134	RPK	6330
DEFINB	16550	RWE	6203
DELETB	17261	SBUFF	6747
DELETE	17433	SETCH	6241
DISPOS	15411	SETCOM	7107
EMPTY	6312	UNPACK	6213
EXTEND	7120	ZADDOM	6711
FETCH	13341	ZGENCOM	6711

C. SAMPLE PROGRAM USING THE COMRADE ASL

In the example that follows, CDMS subroutines SBUFF, ILFN, DEFINB, and FLFN are called to define a data block named JONES. The COMRADE ASL file has been previously attached as LIB.

```
      PROGRAM SAMPLE
      COMMON/AREA/DUM (20000B)
      DIMENSION IBUFF (1025)
      CALL INITLDR (3HLIB, 6HREFILE, IERR)
      IF (IERR.NE.0) STOP 10
      CALL COMRLDR (5HSBUFF, IERR, IBUFF, 1025)
      IF (IERR.NE.0) STOP 20
      CALL COMRLDR (4HILFN, IERR, 6HCDMSDB, 0, 0)
      IF (IERR.NE.0) STOP 30
      CALL COMRLDR (6HDEFINB, IERR, 5HJONES, 8HPERSONAL)
      IF (IERR.NE.0) STOP 40
      CALL COMRLDR (4HFLFN, IERR, 6HCDMSDB)
      IF (IERR.NE.0) STOP 50
      END
```

APPENDIX G

ALPHANUMERIC DATA CONVERSION SUBROUTINES

The library SUBLIB contains a set of subroutines which have been found to be useful to programmers who must manipulate and convert alphanumeric data. The set, known as ALPHAPACK, provides for comparing and moving character strings, and for converting strings to internal representations for octal, integer, real, and double-precision numbers, and vice versa. The ALPHAPACK subroutines are summarized as follows:

SETCH	Sets a given character into an array
MOVE	Moves a character string from one array to another
LOCATE	Searches for the location of one string within another
LSHIFT	Performs a logical shift
CAI	Converts a numeric string to an integer
CIA	Converts an integer to a numeric string
CAR	Converts a numeric string to a real number
CRA	Converts a real number to a numeric string
CAO	Converts an octal string to an octal number
COA	Converts an octal number to an octal string
CAD	Converts a numeric string to a double-precision number
CDA	Converts a double-precision number to a numeric string

ALPHAPACK returns the following error codes:

0 No error detected

1 An illegal character or format encountered

2 Character string contains more characters than allowed

3 The character-positioning parameter is less than 1

4 The number of characters in the string is less than 1

Individual descriptions of the ALPHAPACK subroutines follow.

1. SETCH

SETCH (ARRAY, IL, NBR, CHAR, IERR)

SETCH fills all or part of an array with a specified character.

Parameters:

ARRAY	Name of the array to be filled
IL	First character position of ARRAY to be filled
NBR	Number of character positions to be filled
CHAR	Character (1R format) to be inserted into NBR positions of ARRAY.
IERR	Standard error code

Example:

```
CALL SETCH (ARRAY, 1, 5, IR*, IERR)
```

2. MOVE

MOVE (ARRAY1, IL1, ARRAY2, IL2, NBR, IERR)

MOVE moves a character string from one array to another.

Parameters:

ARRAY1	Name of the array that is to receive the character string
IL1	First character position of ARRAY1 to be filled

ARRAY2 Name of the array containing the character string to be moved.

IL2 First character position of ARRAY2 to be moved.

NBR Number of characters to be moved

IERR Standard error code

Example:

CALL MOVE (ARRAY, 1, ARRAY(4), 6, 15, IERR)

Upon execution of this statement, 15 characters beginning, with the sixth character position of ARRAY(4), will be moved to the first character position of ARRAY(1). FORTRAN word boundaries are not regarded as meaningful when MOVE is executing. An alternate form of the above statement is

CALL MOVE (ARRAY, 1, ARRAY, 36, 15, IERR)

3. LOCATE

LOCATE (ARRAY1, IL1, NBR1, ARRAY2, IL2, NBR2, LOC, IERR)

LOCATE searches for the location of one character string (the one to be matched) within another string (the one to be searched). If a match is realized, the position of the first character of the matching string is returned to the calling program.

Parameters:

ARRAY1 Name of the array containing the string to be matched

IL1 First character position of ARRAY1

NBR1 Number of characters within ARRAY1

ARRAY2 Name of the array containing the string to be searched

IL2 First character position of ARRAY2

NBR2 Number of characters within ARRAY2

LOC Match indicator:

= 0 No match found

= i where i > 0 First character position of the matching string within ARRAY2

IERR Standard error code

Example:

If the contents of the first three words (30 characters) of the array ARRAY were

```
FOUR SCORE AND SEVEN YEARS AGO
1       10        20        30
```

then following execution of the statement

```
CALL LOCATE (5HSEVEN, 1, 5, ARRAY, 1, 30, LOC, IERR)
```

the parameter LOC would be set to 16.

4. LSHIFT

LSHIFT (IWORD, NBITS)

LSHIFT performs left and right logical shifts, i.e., shifts that are neither end-around nor signed-extended. Note that LSHIFT is the only function subprogram described in this manual.

Parameters:

IWORD Word to be shifted

NBITS Indicates the number of bit positions involved and the direction in which the contents of IWORD are to be shifted. If NBITS contains a positive integer, a left logical shift will be performed; if a negative integer, a right logical shift will be

AD-A047 852 DAVID W TAYLOR NAVAL SHIP RESEARCH AND DEVELOPMENT CE--ETC F/G 9/2
COMRADE DATA MANAGEMENT SYSTEM HOST LANGUAGE INTERFACE USERS MA--ETC(U)
JAN 76 W C GORHAM, S E WILLNER, M S SHAW

UNCLASSIFIED DTNSRDC-76-0006 NL

2 OF 2
ADA047852

END
DATE FILMED
1-78
DDC

performed. If NBITS contains a zero, no shift will be performed.

Example:

If the contents of IWORD are 00000...0111000000, the statement

```
N = LSHIFT (IWORD, 2)
```

when executed would result in N having the value of 00000... 011100000000. The statement

```
N = LSHIFT (IWORD, -5)
```

when executed would result in having the value 00000...01110.

5. CAI

CAI (INTEGER, ARRAY, IL, IR, IERR)

CAI converts a numeric string (display code) to an integer. The string may contain no more than 14 characters.

Parameters:

INTEGER	Location to receive the integer
ARRAY	Name of the character string to be converted.
IL	First character position in ARRAY
IR	Last character position in ARRAY
IERR	Standard error code

Example:

If the contents of ARRAY were ABCD12GHIJ, the statement

```
CALL CAI (INT, ARRAY, 5, 6, IERR)
```

when executed would result in INT having the value 12 in fixed format.

6. CIA

CIA (ARRAY, IL, IR, INTEGER, IERR)

CIA converts an integer to a numeric string (display code). The resultant string may contain no more than 15 characters. If the integer is too large for the space provided, an asterisk (*) is placed in the IL position.

Parameters:

ARRAY	Name of the character string
IL	First character position in ARRAY
IR	Last character position in ARRAY
INTEGER	An integer to be converted
IERR	Standard error code

Example:

If the contents of the variable SHIPNAM were the string SHIP, and if the variable SHIPNO were equal to 12, then the statement

```
CALL CIA (SHIPNAM, 5, 6, SHIPNO, IERR)
```

when executed would result in SHIPNAM containing the string SHIP12.

7. CAR

CAR (REAL, ARRAY, IL, IR, IERR)

CAR converts a numeric string (display code) to a real number. The string may contain no more than 20 characters.

Parameters:

REAL	Location to receive the real number

ARRAY Name of the character string to be converted. The string may not contain embedded blanks or non-numeric characters other than E, +, . and -.

IL First character position in ARRAY

IR Last character position in ARRAY

IERR Standard error code

Example:

If the contents of the variable ARRAY were ABC5237.00, the statement

CALL CAR (REAL, ARRAY, 4, 10, IERR)

when executed would result in REAL having the contents 5237.00 in floating format.

8. CRA

CRA (ARRAY, IL, FORMAT, REAL, IERR)

CRA converts a real number to a numeric string according to a user-specified conversion format. The resultant string may contain no more than 20 characters. No format checking is performed; legal formats for real numbers include the E and F formats described in the FORTRAN Reference Manual.[11]

Parameters:

ARRAY Name of the character string

IL First character position in ARRAY

FORMAT Name of the variable containing the conversion format (left-justified, H format); e.g., 8H(E20.14)

REAL A real number to be converted

IERR Standard error code

9. CAO

CAO (OCTAL, ARRAY, IL, IR, IERR)

CAO converts a numeric string (display code) to an octal number. The string may not contain more than 20 characters.

Parameters:

OCTAL	Location to receive the octal number
ARRAY	Name of the character string to be converted.
IL	First character position in ARRAY
IR	Last character position in ARRAY
IERR	Standard error code

Example:

If the contents of the variable ARRAY were ABC123, the statement

CALL CAO (IOCT, ARRAY, 4, 6, IERR)

when executed would result in IOCT having the contents 83 (i.e., 123 decimal) in fixed format.

10. COA

COA (ARRAY, IL, IR, IOCT, IERR)

COA converts an octal number to a numeric string. If the number is too large for the string specified, an asterisk (*) is placed in the IL position.

Parameters:

ARRAY	Name of the character string
IL	First character position in ARRAY
IR	Last character position in ARRAY
IOCT	An octal number to be converted
IERR	Standard error code

11. CAD

CAD (DOPR, ARRAY, IL, IR, IERR)

CAD converts a numeric string (display code) to a double-precision real number. The string may contain no more than 30 characters.

Parameters:

DOPR	Location to receive the double-precision number.
ARRAY	Name of the character string to be converted. The string may not contain embedded blanks or non-numeric characters other than D, +, . and -.
IL	First character position in ARRAY
IR	Last character position in ARRAY
IERR	Standard error code

12. CDA

CDA (ARRAY, IL, FORMAT, DOPR, IERR)

CDA converts a double-precision real number to a numeric string according to a user-specified conversion format. The resultant string may contain no more than 30 characters. No format checking is performed; the legal format for double-precision numbers is described in the FORTRAN Reference Manual.[11]

Parameters:

ARRAY	Name of the character string
IL	First character position in ARRAY
FORMAT	Name of the variable containing the conversion format (left-justified, H format); e.g., 8H(D28.22)

DOPR	A double-precision real number to be converted
IERR	Standard error code

APPENDIX H

PERMANENT FILE MANAGEMENT SUBROUTINES

When an application program is executing, especially one which relies heavily on input and output processing and file handling, it may become necessary for the program to dynamically perform file functions ordinarily accomplished by way of job control statements. Towards this end, the library SUBLIB provides two sets of file management subroutines, one set for files managed by the SCOPE system, and one set for files managed by the COMRADE Permanent File Management System (CPFMS; see Appendix A). Parameters, error codes, and calling formats for a subset of the subroutines follow. Detailed descriptions of all the subroutines are contained in the COMRADE Design Administration System Users Manual.[13]

A. SCOPE FILE MANAGEMENT SUBROUTINES

1. Parameter Identification

IERR	Error Code
LFN	Local File Name
IPFN	Permanent file name (up to 40 characters)
ID	User identifier (4 characters)
ITK	Turnkey password (up to 9 characters)
IRD	Read password (up to 9 characters)
IEX	Extend password (up to 9 characters)
IMD	Modify password (up to 9 characters)
ICN	Control password (up to 9 characters)

IMR	Read/Write indicator:
	= 0 Read/write
	≠ 0 Read only
IAC	Account number (10 characters)
ICY	Cycle number:
	= 0 Highest cycle will be processed
	= i where 1 ≤ i ≤ 999 Cycle i will be processed
IRP	Retention period in days (must be ≤ 999; default is 30)

2. Error Codes

0	No error detected
1	IAC Illegal
2	LFN already in use
3	Unknown LFN
4	No room for extra cycle (limit is 5)
5	Permanent file catalog full
6	No LFN or PFN
8	Latest index not written
9	File not on a permanent file device
10	File not in system
12	Invalid cycle number
13	Cycle number limit reached
14	Permanent file directory full
15	Function attempted on non-permanent file
16	Function attempted on non-local file
18	File never assigned to a device

20	Permanent file already attached
21	File unavailable

3. Calling Formats

The calling formats for subroutines ATTACH, CATALOG, and PURGE are shown below; these subroutines may be used by an executing program to attach, catalog, and purge SCOPE permanent files. The calling formats for the subroutines are nearly identical. For subroutines ATTACH and PURGE, the calling formats may be truncated to include only the required parameters IERR, LFN, IPFN, and ID; the calling format for CATLOG may be truncated to include the required parameters IERR, LFN, IPFN, ID, ITK, IRD, IEX, IMD, ICN, IMR, and IAC. In any case, unused parameters should be set to 0.

```
CALL ATTACH (IERR, LFN, IPFN, ID, ITK, IRD, IEX, IMD, ICN,
    IMR, IAC, ICY, IRP)
CALL CATLOG (same as ATTACH)
CALL PURGE (same as ATTACH)
```

B. CPFMS FILE MANAGEMENT SUBROUTINES

1. Parameter Identification

IERR	Error code
LFN	Local file name
L3	Level-3 name (as many as 8 characters)
L4	Level-4 name (as many as 8 characters)
IFACF	File access control field of 7 bits:
	= 0 check only file access key (FAK)
	with the file access lock (FAL)

IMR	Read/write indicator:
	= 0 Read only
	= 1 Read/write
IPSPW	Pseudo-password (as many as 5 characters)
	= 0 No password
IFAL	File access lock of 20 bits
	= 0 No lock

2. Error codes

0	No error detected
1	Invalid pseudo-password
2	LFN already in use
3	Access for file creator only
4	FAK does not agree with FAL
5	Level 3 name does not exist
6	Level 4 name does not exist
7	LFN already in list of attached files
8	LFN table full
10	File busy
15	Permanent file catalog full
16	Latest index not written
17	File not on a permanent file device
23	Permanent file already attached

3. Calling Formats

The calling formats for subroutines COMATC, COMCAT, and COMPRG are shown; these subroutines may be used by an executing program to attach, catalog, and purge CPFMS permanent files.

```
CALL COMATC (IERR, LFN, L3, L4, IMR, IPSPW)
CALL COMCAT (IERR, LFN, L3, L4, IFACF, IPSPW, IFAL)
CALL COMPRG (IERR, LFN, L3, L4, IPSPW)
```

REFERENCES

1. Rhodes, T., and W. Gorham, "COMRADE - The Computer-Aided Design Environment Project - An Introduction," DTNSRDC Report 76-0001 (Jan 1976).

2. "Naval Ship Systems Command Technical Development Plan-Computer-Aided Ship Design and Construction," NAVSHIPS Report S46-33X (Feb 1970).

3. Control Data Corporation, "SCOPE Reference Manual, 6000 Version 3.4," Publication No. 60307200.

4. Wallace, M. et al., "COMRADE Data Storage Facility Users Manual," DTNSRDC Report 76-0003 (Jan 1976).

5. Gorham, W. et al., "COMRADE Data Management System Conversational Interface Users Manual," DTNSRDC Report 76-0007 (Jan 1976).

6. Control Data Corporation, "INTERCOM Reference Manual, 6000 Version 4," Publication No. 60307100.

7. CODASYL Systems Committee, "Feature Analysis of Generalized Data Base Management Systems," CODASYL Systems Committee Report (May 1971).

8. Fong, E. et al., "Six Data Base Management Systems: Feature Analysis and User Experience," National Bureau of Standards Technical Note 887 (Nov 1975).

9. Wallace, M., "Absolute Subroutine Utility Users Manual," DTNSRDC Report 76-0004 (Jan 1976).

10. Control Data Corporation, "COBOL Version 4 Reference Manual," Publication No. 60384100.

11. Control Data Corporation, "FORTRAN Extended Version 4 Reference Manual," Publication No. 60305601.

12. Control Data Corporation, "COMPASS Version 3 Reference Manual," Publication No. 60360900.

13. Chernick, C.M., "COMRADE Design Administration System Users Manual," DTNSRDC Report 76-0008.

INITIAL DISTRIBUTION

Copies	
1	U.S. Army Electronics Command/AMSEL-GG-CG Robert Dunn
1	CHONR 437/Denicoff
1	NAVSEA 6E24/Chaiken
25	NAVSEC: 1 SEC 6105B/Dietrich 20 SEC 6105B/Berry 1 SEC 6114B2/Fuller 1 SEC 6133E/Straubinger 1 SEC 6152D/Lee 1 SEC 6179A20/Singer
12	DDC
1	NASA-Langley/R. Fulton
1	Grumman Aerospace Corp./William H. Mueller

CENTER DISTRIBUTION

Copies		Copies	
1	18/1808 Gleissner	1	185/Corin
1	1802.2 Frankiel	1	1851/Brainin
1	1802.3/Theilheimer	1	1853/Thomson
1	1805/Cuthill	1	1854/Lynch
2	1809/Harris	1	1855/Brengs
1	182/Camara	1	1856/Skall
1	1822/Rhodes	1	189/Gray
1	1824/Zaritsky	1	1892.1/Strickland
1	1826/Culpepper	30	5211 Reports Distribution
20	1828/Gorham	1	5221 Library - Carderock
1	184/Lugt	1	5222 Library - Annapolis
		1	712.9/Pierce

DTNSRDC ISSUES THREE TYPES OF REPORTS

1. DTNSRDC REPORTS, A FORMAL SERIES, CONTAIN INFORMATION OF PERMANENT TECHNICAL VALUE. THEY CARRY A CONSECUTIVE NUMERICAL IDENTIFICATION REGARDLESS OF THEIR CLASSIFICATION OR THE ORIGINATING DEPARTMENT.

2. DEPARTMENTAL REPORTS, A SEMIFORMAL SERIES, CONTAIN INFORMATION OF A PRELIMINARY, TEMPORARY, OR PROPRIETARY NATURE OR OF LIMITED INTEREST OR SIGNIFICANCE. THEY CARRY A DEPARTMENTAL ALPHANUMERICAL IDENTIFICATION.

3. TECHNICAL MEMORANDA, AN INFORMAL SERIES, CONTAIN TECHNICAL DOCUMENTATION OF LIMITED USE AND INTEREST. THEY ARE PRIMARILY WORKING PAPERS INTENDED FOR INTERNAL USE. THEY CARRY AN IDENTIFYING NUMBER WHICH INDICATES THEIR TYPE AND THE NUMERICAL CODE OF THE ORIGINATING DEPARTMENT. ANY DISTRIBUTION OUTSIDE DTNSRDC MUST BE APPROVED BY THE HEAD OF THE ORIGINATING DEPARTMENT ON A CASE-BY-CASE BASIS.